Geometria Computacional

Professor:

Anselmo Montenegro
www.ic.uff.br/~anselmo

Conteúdo:

- Problemas fundamentais

Roteiro

- Introdução
- Problemas de localização
- Interseção de segmentos
- Problema do par mais próximo
- Problema do círculo envolvente mínimo

Problemas fundamentais: introdução

- Iremos apresentar alguns dos problemas mais básicos em Geometria Computacional
- Tais problemas servem em alguns casos como base para solução de outros problemas mais complexos

Problemas fundamentais: localização de pontos

- Os problemas de localização de pontos em relação a diferentes primitivas são fundamentais para solução de diversos problemas em GC

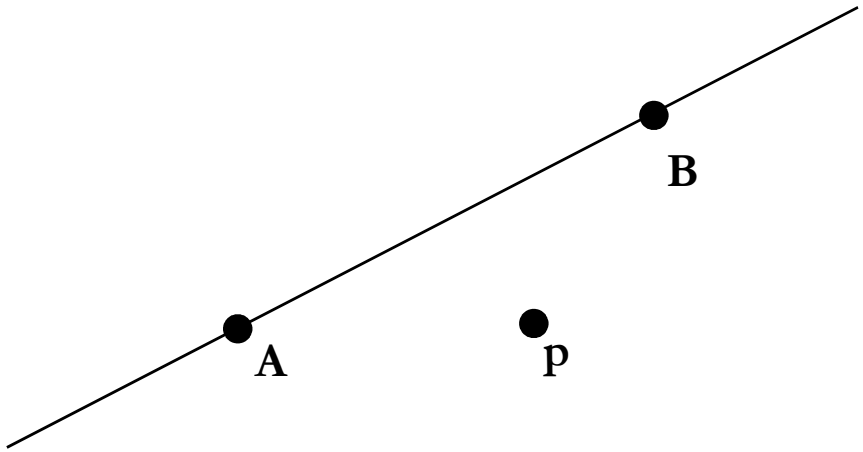
- Consideramos aqui três versões do problema:

  - Localização de pontos em relação a retas
  - Localização de pontos em relação a triângulos
  - Localização de pontos em relação a polígonos

Problemas fundamentais: localização de pontos em relação a retas

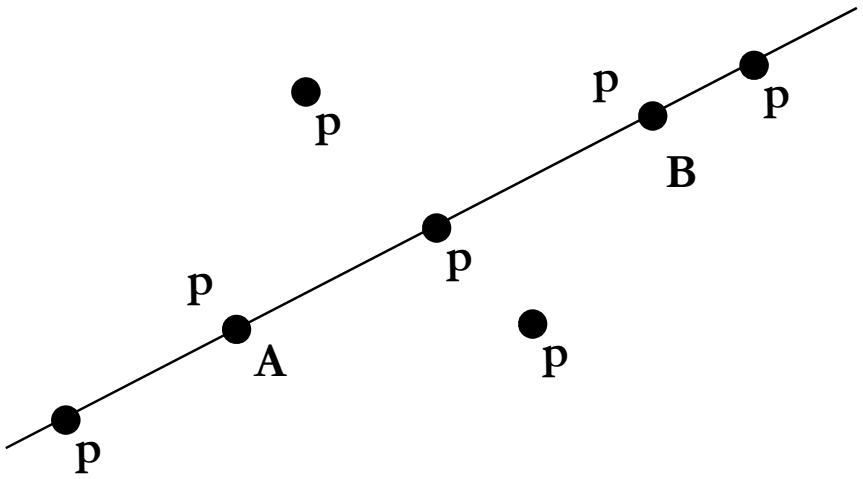
- Problema : dada uma reta determinada por dois pontos A e B, orientada no sentido de A para B, determinar se um ponto p está sobre a reta, no semi-plano direito ou semi-plano esquerdo

- Uma reta divide o plano em dois semi-planos ilimitados



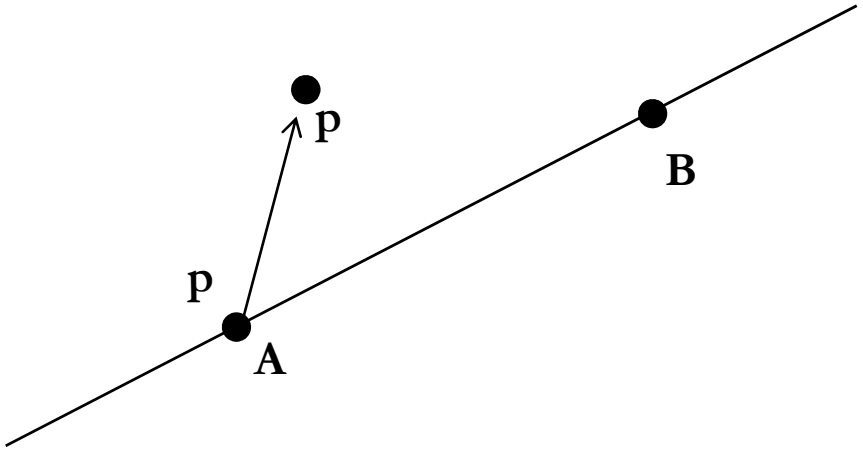
Problemas fundamentais: localização de pontos em relação a retas

- Quando p está sobre a reta determinada por A e B pode-se distinguir 5 casos:
  - p está antes de A
  - p está sobre A
  - p está entre A e B
  - p está sobre B
  - p está após B



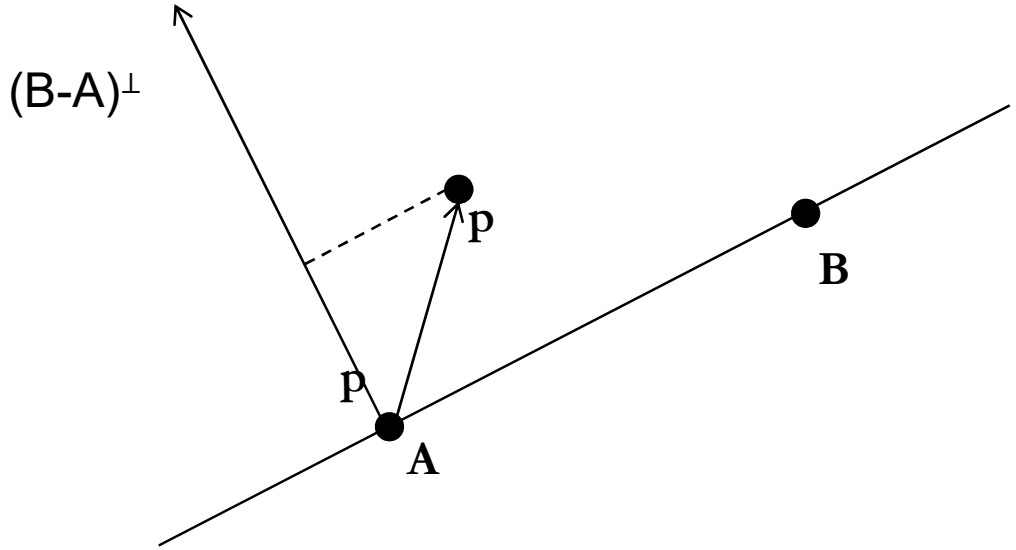
Problemas fundamentais: localização de pontos em relação a retas

- p está a esquerda do segmento orientado AB se o vetor (p-A) tem mesma direção que uma rotação positiva do vetor (B-A)



Problemas fundamentais: localização de pontos em relação a retas

- Em outras palavras, a componente de (p-A) em relação ao complemento ortogonal positivo $(B-A)^{\perp} = (-(By-Ay), Bx-Ax)$ é positiva



- Logo, p está a esquerda se $\left\langle (p-A), (B-A)^{\perp} \right\rangle > 0$

Problemas fundamentais: localização de pontos em relação a retas

- Exercício: mostre que

$$\left\langle (p-A), (B-A)^{\perp} \right\rangle = \begin{vmatrix} 1 & 1 & 1 \\ x_A & x_B & x_p \\ y_A & y_B & y_p \end{vmatrix}$$

Problemas fundamentais: localização de pontos em relação a retas

- O resultado anterior mostra que p está à esquerda da reta que passa por AB no sentido de A para B se:

$$(B-A)\times(p-A)>0$$

- De modo análogo modo p está à direita se

$$(B-A)\times(p-A)<0$$

Problemas fundamentais: localização de pontos em relação a retas

- No caso degenerado em que $(B-A)\times(p-A)=0$, p está sobre a reta

- Se o segmento AB não for vertical podemos utilizar as seguintes regras

Condição	**Localização**
$X_p<X_A$	p está antes de A
$X_p=X_A$	p está em A
$X_A<X_p<X_B$	p está entre A e B
$X_p=X_B$	p está em B
$X_B<X_p$	p está depois de B

Problemas fundamentais: localização de pontos em relação a retas

- Exercício: como especificar regras para o caso em que AB é vertical

Problemas fundamentais: localização de ponto em triângulo

- Um problema muito importante em geometria computacional é localizar pontos em polígonos. Comecemos com o problema mais simples, de localizar um ponto em um triângulo

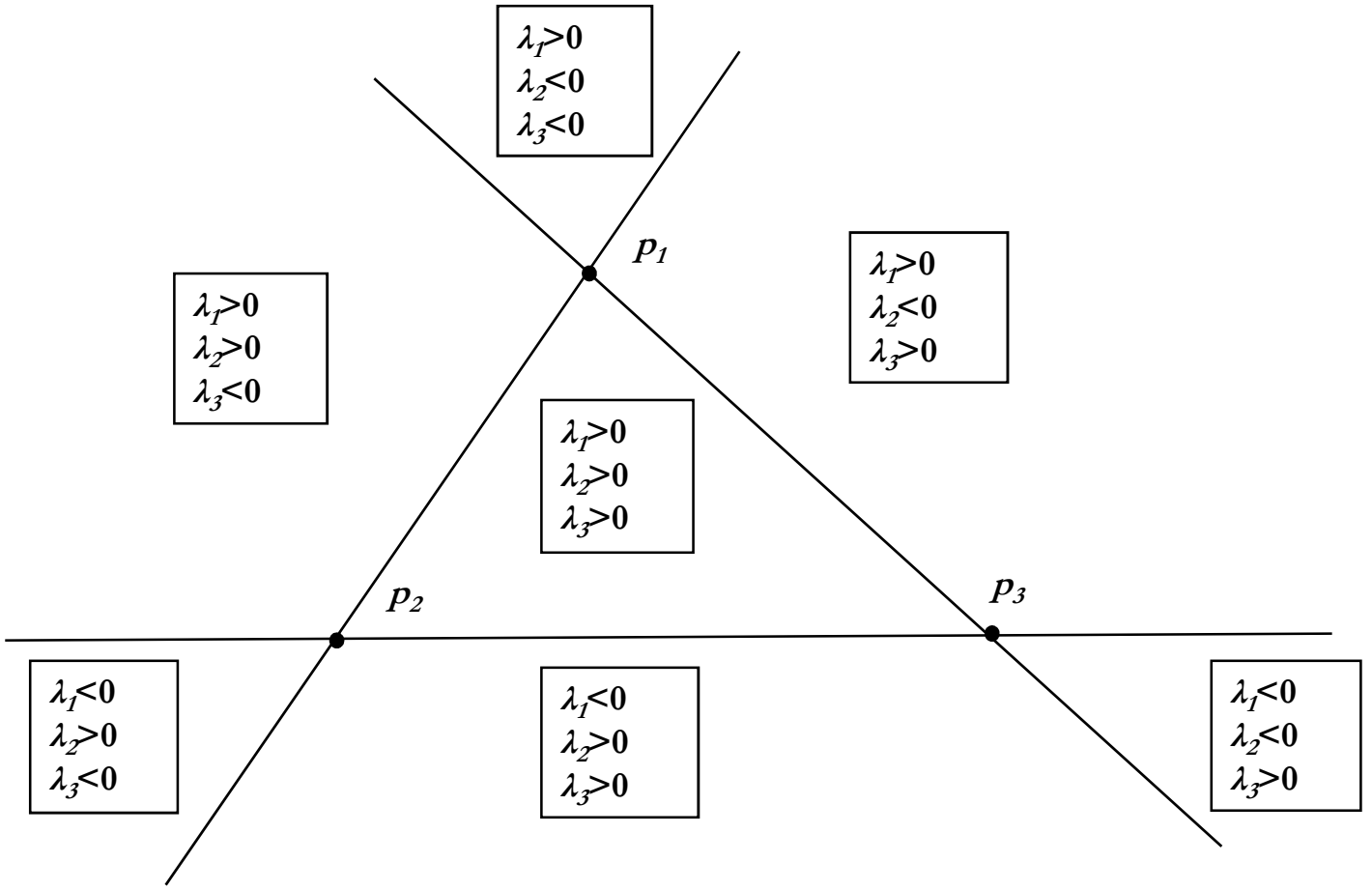
- *Teorema 5.1* – Sejam p_1, p_2 e p_3 três pontos em posições gerais (não colineares) no R^2. Então cada ponto p do plano pode ser escrito como

$$p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$$

- onde $\lambda_1 + \lambda_2 + \lambda_3 = 1$

- Os coeficientes λ_1, λ *e* λ_3 são conhecidos como coordenadas baricêntricas.

Problemas fundamentais:localização de ponto em triângulo



Sinais das coordenadas baricêntricas

Problemas fundamentais: localização de ponto em triângulo

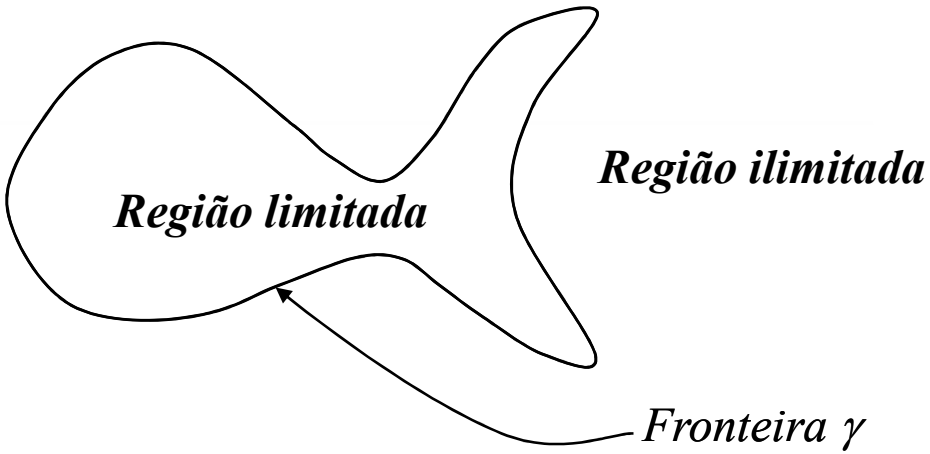
- Exercício: prove o teorema 5.1
- Exercício: implemente um algoritmo para localização de ponto em um triângulo

Problemas fundamentais: localização de pontos em relação a poligonos

- Problema (Ponto em polígono) – Dado um polígono simples $P = p_1, p_{2,...,} p_n$ *no* R^2, determinar se um ponto *p* é interior, exterior ou encontra-se na fronteira de *P*

- *Teorema 5.2 (Teorema de Jordan)* – uma curva poligonal simples separa o plano em duas regiões conexas abertas, uma limitada e a outra não, ambas tendo a curva como fronteira

Problemas fundamentais: localização de pontos em relação a poligonos

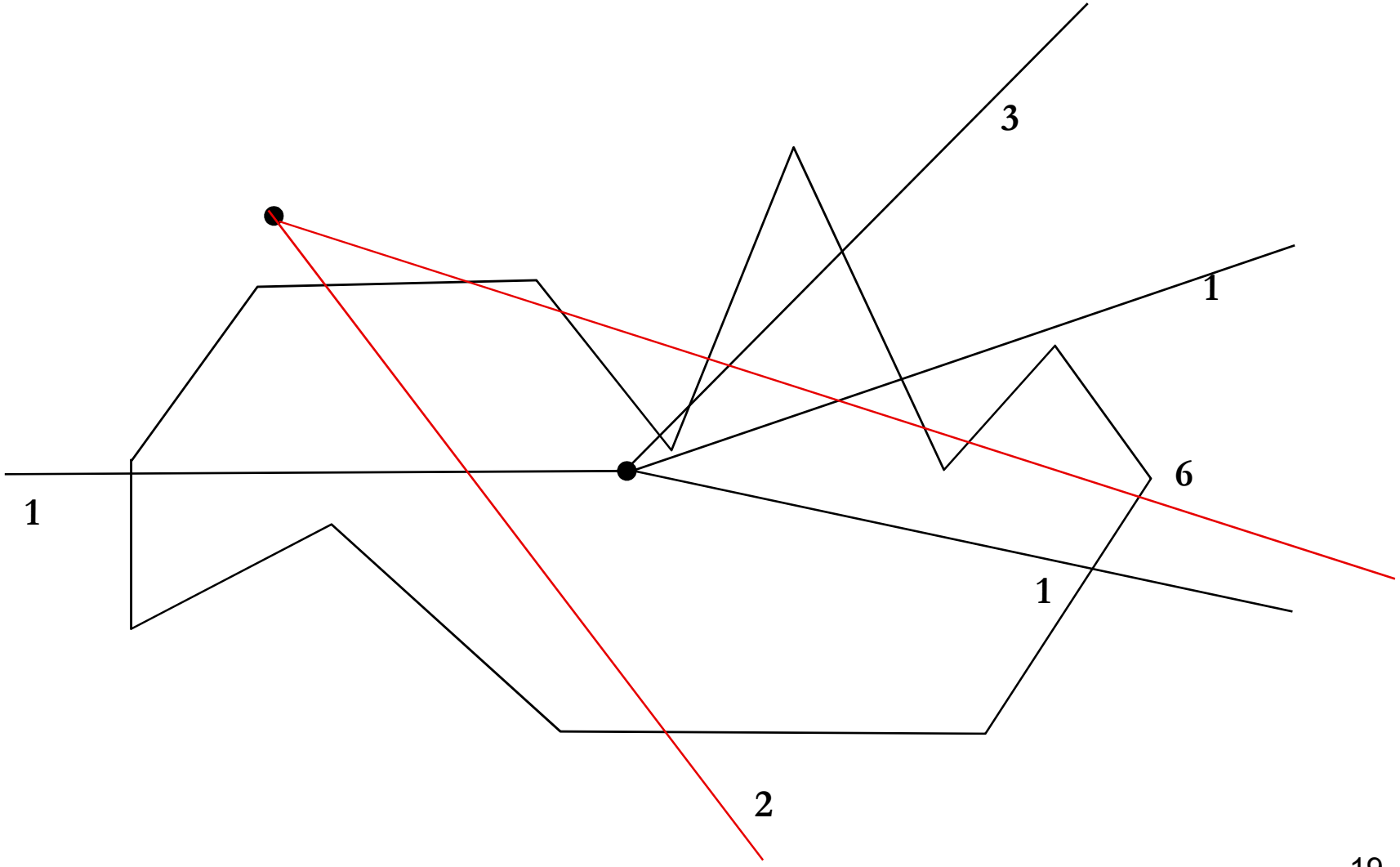


Problemas fundamentais:localização de pontos em relação a poligonos

- Solução baseada no Teorema de Jordan:
  - Traçar uma semi-reta *L* partindo de *p* e determinar os pontos de interseção com a linha poligonal *P*

Configuração	**Resposta**
p em uma aresta	p está na fronteira de P
#interseções par	p é exterior a P
#interseções ímpar	p é interior a P

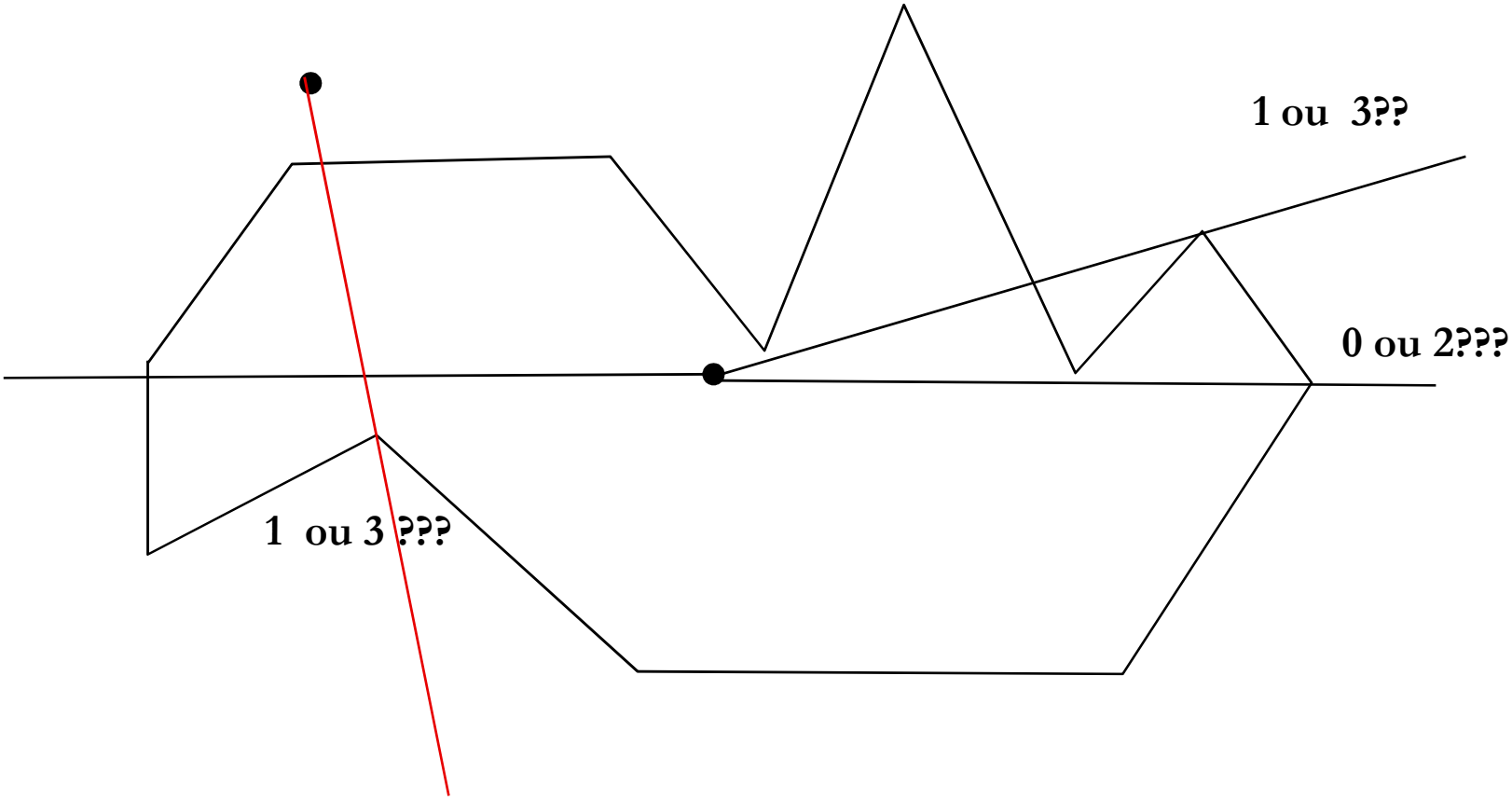
Problemas fundamentais: localização de pontos em relação a poligonos



Problemas fundamentais: localização de pontos em relação a poligonos

- Exercício: explique a lógica por trás do teste de paridade.

Problemas fundamentais: localização de pontos em relação a poligonos



Problemas fundamentais: localização de pontos em relação a poligonos

- Como resolver esse problema de interseção nos vértices??

- A interseção de um lado $p_i p_{i+1}$ com L é contada somente se ela ocorrer em um ponto que não de seja de ordenada mínima em $p_i p_{i+1}$

- Exercício: explique porque esta estratégia funciona

Problemas fundamentais: localização de pontos em relação a poligonos

- Como resolver esse problema de interseção nos vértices??

- A interseção de um lado $p_i p_{i+1}$ com L é contada somente se ela ocorrer em um ponto que não de seja de ordenada mínima em $p_i p_{i+1}$

- Exercício: explique porque esta estratégia funciona

Problemas fundamentais: localização de pontos em relação a poligonos

- Exercício: implemente um algoritmo que faça a detecção ponto polígono

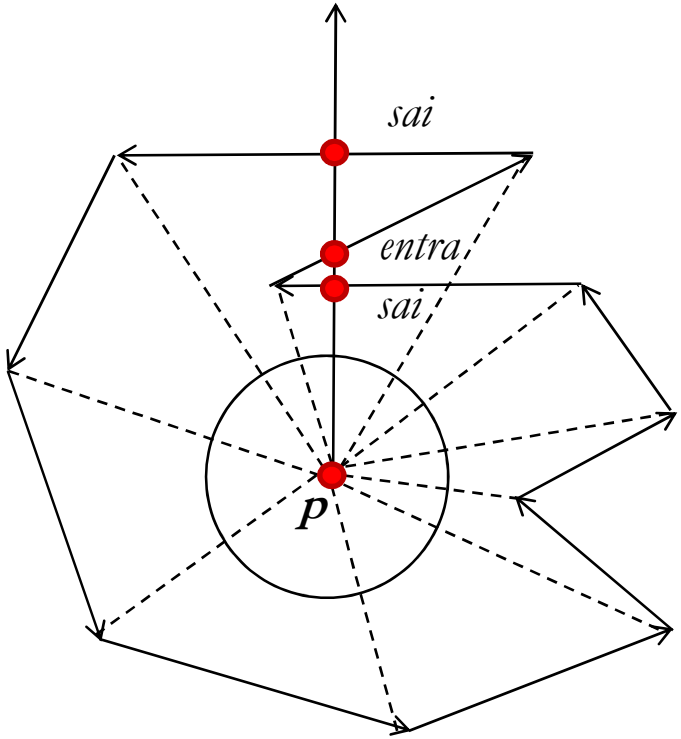
Problemas fundamentais: índices de rotação

- Dada uma linha poligonal $L = p_1, p_{2,\ldots,} p_n$ e um ponto p não pertencente a ela, o índice de rotação de p em relação a L é dado por

$$\kappa = \frac{1}{2\pi} \sum_{i=1}^{n} \angle p_i p p_{i+1}$$

Problemas fundamentais: índices de rotação

- Dada uma linha poligonal $L = p_1, p_2, \ldots, p_n$ e um ponto p não pertencente a ela, o índice de rotação de p em relação a L é dado por



Problemas fundamentais: índices de rotação

- *Teorema 5.3 – Seja P = um polígono simples e seja p um ponto que não pertence à fronteira de P. Então o índice de rotação κ de p é igual a 0,1 ou -1 e*

  a) p é interior se e somente se κ=1 ou -1

  b) p é exterior se e somente se κ=0

Problemas fundamentais: índices de rotação

- *Exercício: provar o teorema 2.5. Dica: considere a contribuição de cada setor gerado pelos segmentos com origem em p e destino nos vértices de P com base no número de vezes que cada lado de P cruza cada* setor

Problemas fundamentais: interseção de segmentos

- Considere uma configuração com *n* segmentos de reta no plano

  Problema 1. Determinar se existe um par de segmentos que se cruzam (detecção)

  Problema 2. Determinar todos os pares de segmentos que se cruzam (identificação)

Problemas fundamentais: interseção entre 2 segmentos - decisão

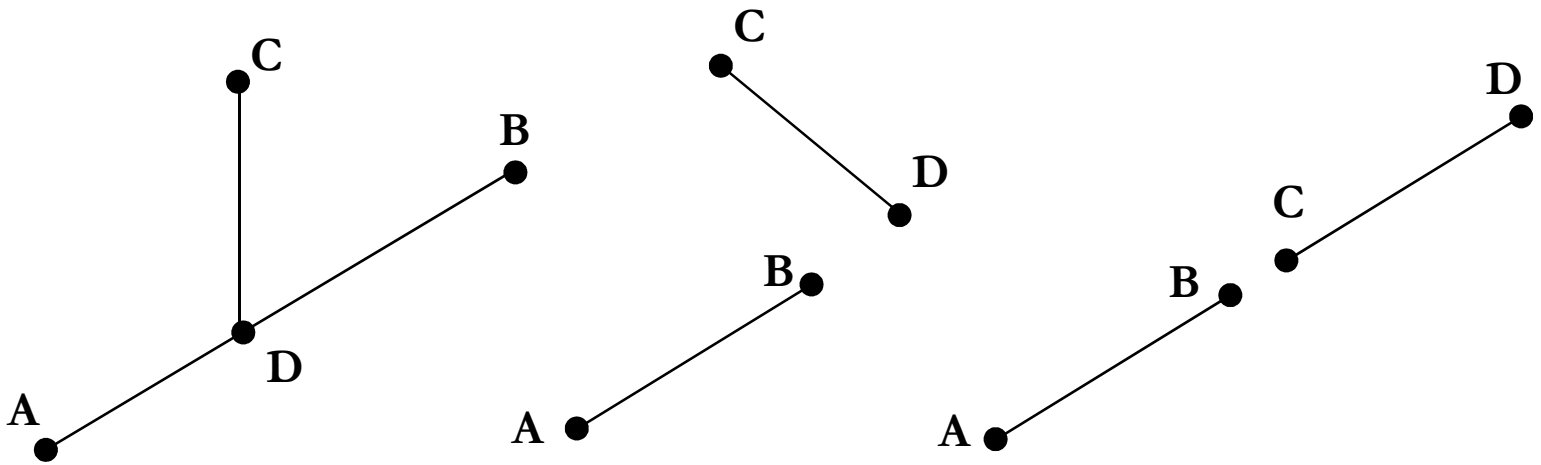
- Sejam AB e CD dois segmentos.

- Solução do problema de decisão: AB e CD se cruzam se C e D estão em lados opostos em relação a AB e se A e B estão em lados opostos em relação a CD

Problemas fundamentais: interseção entre 2 segmentos - decisão

- Exercício: expresse o critério anterior através das operações primitivas estabelecidas

- Exercício: interprete a noção de lados opostos para configurações degeneradas



Problemas fundamentais: interseção entre 2 segmentos - identificação

- A solução para o problema de identificação do ponto de interseção de dois segmentos AB e CD pode ser obtida através da solução de um sistema linear

- Primeiramente encontramos o ponto de interseção p entre AB e CD, caso exista, e verificamos se p é interior a ambos os segmentos

Problemas fundamentais: interseção entre 2 segmentos - identificação

- Seja *r* a reta que passa por AB e parametrizada por A+t(B-A) e *s* a reta que passa por CD e parametrizada por C+t(C-D), $t,s \in R$

- Pode se afirmar que a interseção quando existe é dada por p = A+t(B-A) = C+t(C-D), $t,s \in [0,1]$

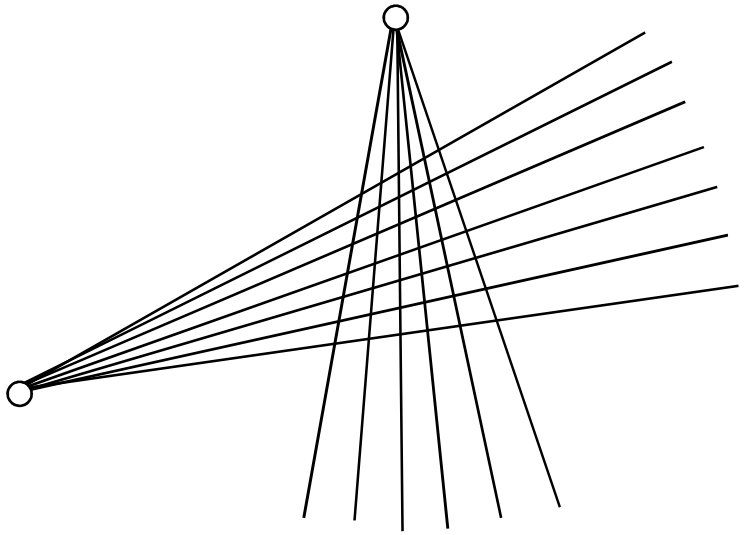
$$A_x + t(B_x - A_x) = C_x + s(D_x - C_x)$$
$$A_y + t(B_y - A_y) = C_y + s(D_y - C_y)$$

$$\begin{pmatrix} B_x & A_x \\ B_y & A_y \end{pmatrix} \begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} C_x - A_x \\ C_y - A_y \end{pmatrix}$$

Problemas fundamentais: interseção entre n segmentos

- O problema geral de interseção de *n* segmentos em suas duas versões (detecção e identificação) admite uma solução força bruta: **Testar todos os pares**

- A solução força bruta entretanto é O(n^2)



Problemas fundamentais: interseção entre n segmentos

- Exercício: mostre porque a solução força bruta é $O(n^2)$

Problemas fundamentais: interseção entre n segmentos

- É possível fazer melhor?
- Sim. O limite inferior para identificação não se aplica a detecção
- Para detecção temos apenas o limite inferior trivial Ω(n)

Problemas fundamentais: interseção entre n segmentos

- Outra visão é a de que o número de cruzamentos I pode ser muito menor que O (n^2)
- Logo é razoável considerar a complexidade de identificação em função de I (em função do tamanho da saída do problema)
- Assim temo um limite inferior de Ω(n+I) para identificação

Problemas fundamentais: interseção entre n segmentos

- Como estabelecer um limite inferior não trivial para o problema?
- Usar a técnica de redução
- Infelizmente, ordenação não se reduz a identificação ou detecção

Problemas fundamentais: interseção entre n segmentos

- É possível entretanto reduzir um outro problema muito simples ao problema de identificação

- **Problema (Unicidade)** – dados *n* números reais $x_1,x_2,...,x_n$, decidir se há duplicatas entre eles, isto é, se existem *i* e *j* com $i \neq j$

- Solução: ordenar os *n* números e verificar se há dois consecutivos iguais

Problemas fundamentais: interseção entre n segmentos

- Unicidade $\propto$Ordenação $\Rightarrow$O(n*log*n)
- Observe que Unicidade $\propto$Ordenação $\neg\Rightarrow$ Ω(n*log*n)
- Para isso teríamos que mostrar que: Ordenação $\propto$Unicidade

Problemas fundamentais: interseção entre n segmentos

- Ordenação ∝Unicidade não é conhecida
- Porém sabe-se que Unicidade é Ω(n*log*n)

> **No modelo de árvores de decisões algébricas, qualquer algoritmo que decida se há duplicatas numa sequência de n números leva tempo** Ω(n*log*n)

Problemas fundamentais: interseção entre n segmentos

- Unicidade ∝Detecção – construa para cada número real x_i em $x_1,x_2,...,x_n$, um segmento de reta $s_i = \{x_i\} \times [0,1]$.

- $x_i = x_j \Leftrightarrow s_i \cap s_j \neq \emptyset$

- Logo detecção é $\Omega(n\log n)$

Problemas fundamentais: interseção entre n segmentos

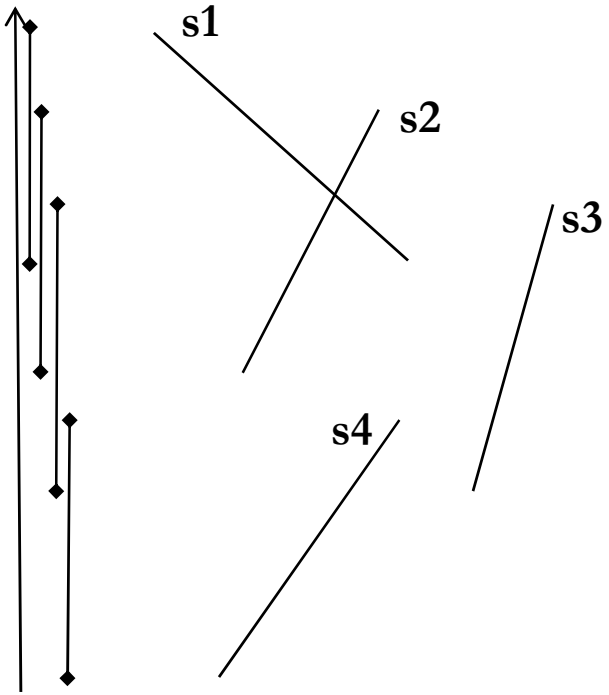
- É possível reduzir detecção a identificação (como ?)
- Logo, podemos transferir o limite inferior $\Omega(n\log n)$ de detecção para identificação
- Tal limite ($\Omega(n\log n)$) não é o melhor pois sabemos que identificação é $\Omega(n^2)$
- Por outro lado, considerando o número de cruzamentos I podemos chegar ao limite inferior $\Omega(n\log n + I)$

Problemas fundamentais: algoritmo para interseção entre n segmentos

- Como evitar efetuar o teste entre todos os n^2 pares de segmentos em conjunto de segmentos S?
- O segredo é usar a característica geométrica do problema:
- Segmentos que estão longe são improváveis de produzirem uma interseção, diferentemente do que ocorre com os que estão próximos

Problemas fundamentais: algoritmo para interseção entre n segmentos

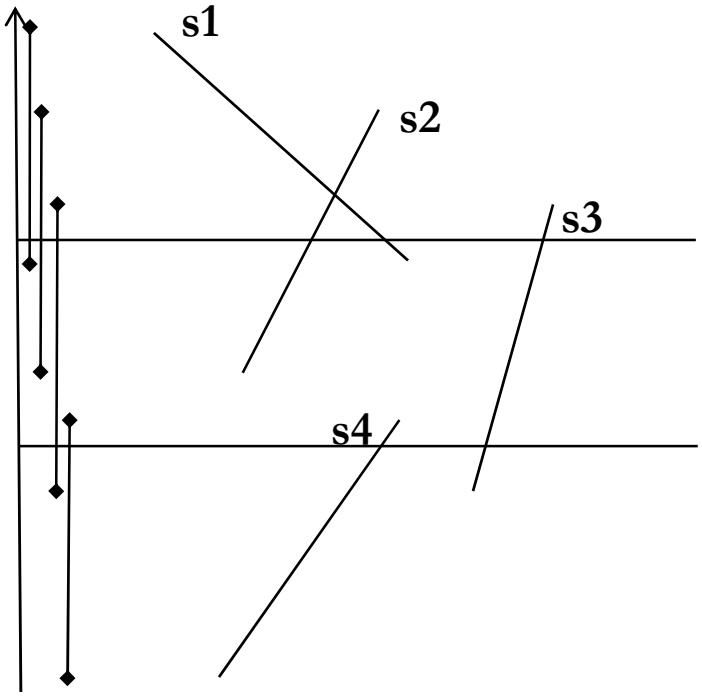
- Observação 1: dois segmentos s_i e s_j podem se intersectar somente se a projeção das coordenadas dos seus extremos no eixo *y* determina intervalos com alguma sobreposição



Podemos no exemplo a esquerda descartar a possibilidade de interseção que o segmento s4 intersecte os segmentos s1 e s2

Problemas fundamentais: algoritmo para interseção entre n segmentos

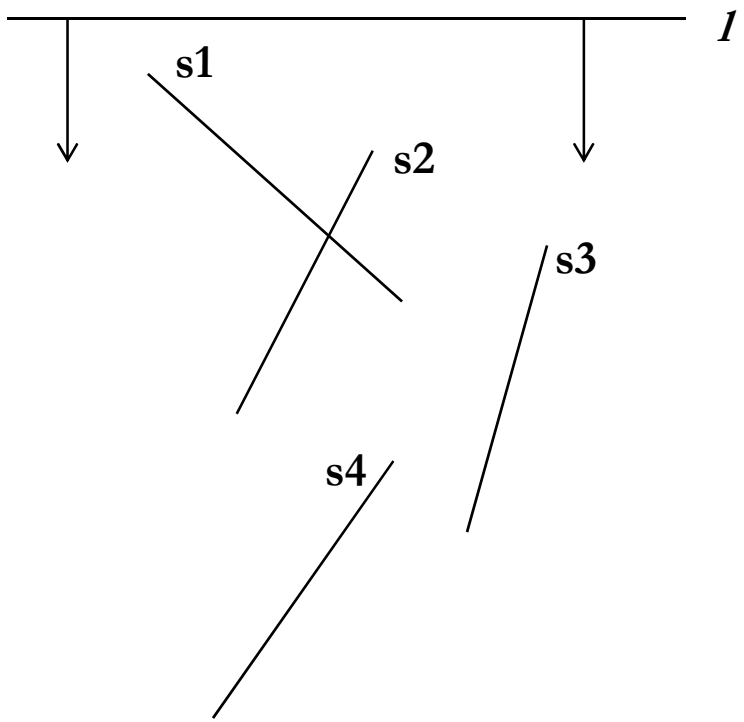
- Observação 2: um par de segmentos s_l e s_r podem se intersectar se existe uma reta l horizontal que os intersecta



Podemos então no exemplo a esquerda descartar a possibilidade de interseção entre os pares $\{s_1, s_4\}$ e $\{s_2, s_4\}$

Problemas fundamentais: algoritmo para interseção entre n segmentos

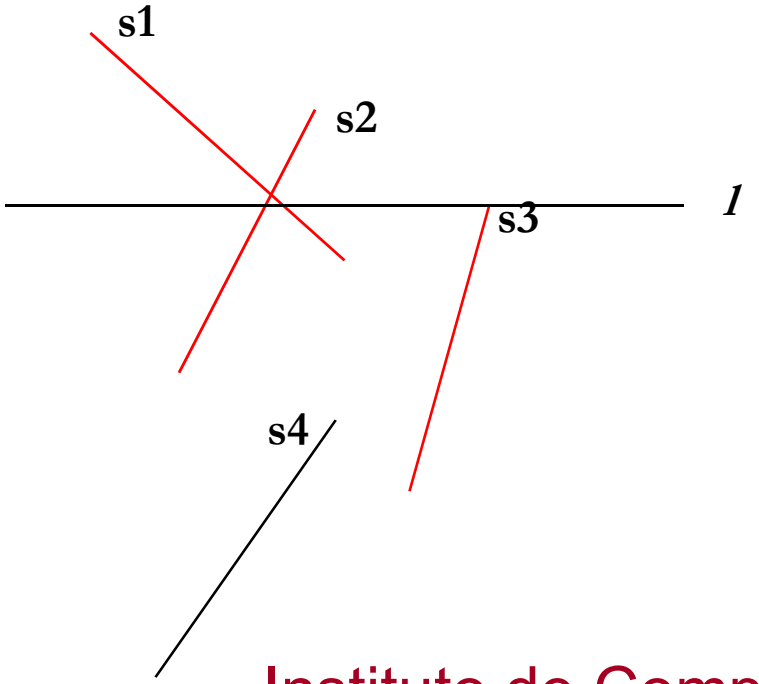
- A ideia para resolver o problema consiste em usar uma reta *l* que varre o plano
- *l* começa de uma posição acima de todos os segmentos e armazena, a cada instante, quais são os segmentos que a intersectam



Problemas fundamentais: algoritmo para interseção entre n segmentos

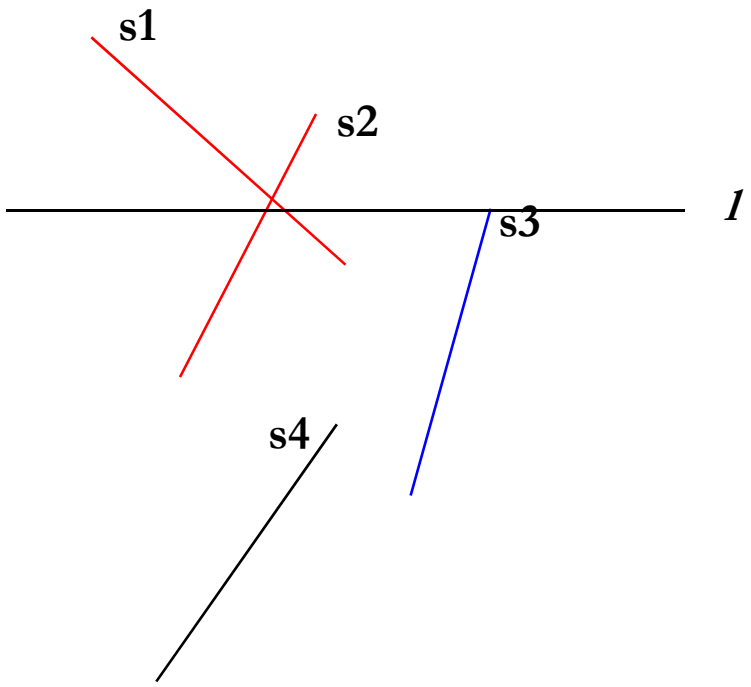
- Um algoritmo assim construído é baseado em uma técnica denominada *varredura do plano.* A reta *l* é denominada *reta de varredura*

- O *status* ou *estado* da reta é conjunto de segmentos que a intersectam



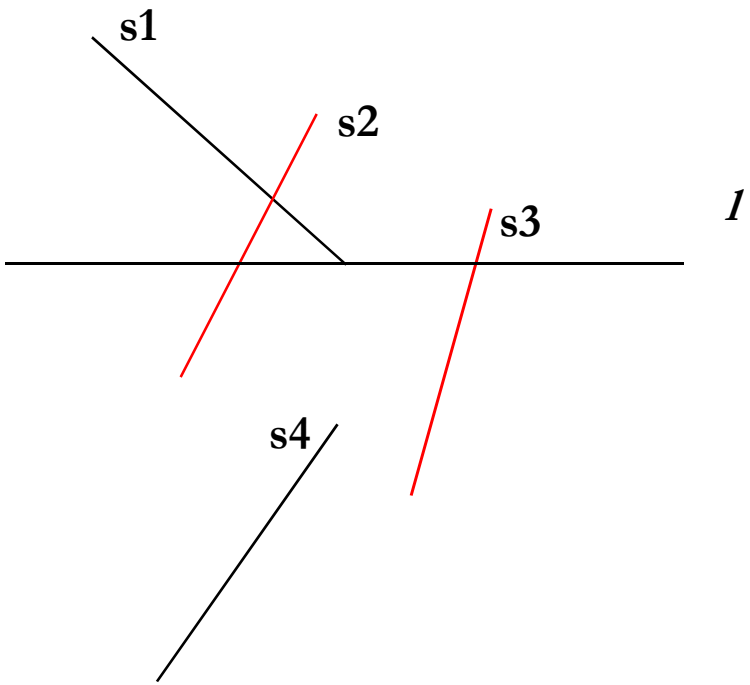
Problemas fundamentais: algoritmo para interseção entre n segmentos

- O status muda conforme a reta desce, mas não continuamente



Problemas fundamentais: algoritmo para interseção entre n segmentos

- O status muda conforme a reta desce, mas não continuamente



Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- O estado é modificado em pontos especiais denominados *pontos de eventos*
- Os pontos extremos dos segmentos formam o conjunto de eventos
- Veremos posteriormente que pontos de interseção computados também são eventos

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- O algoritmo apenas efetua ações quando os eventos são atingidos
- Ações realizadas:
  - Atualizar o status da reta de varredura
  - Calcular interseções

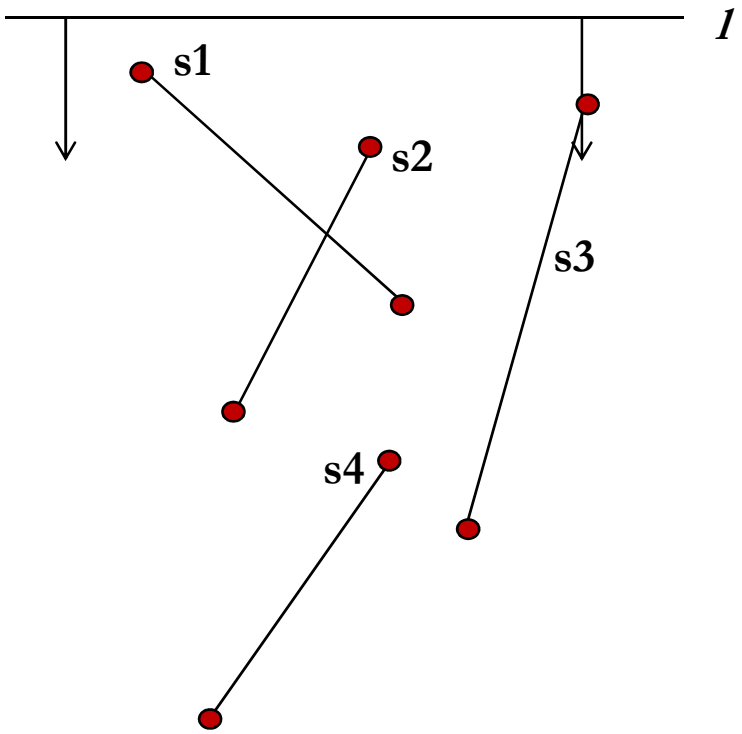
Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Caso 1 (o evento é um ponto extremo superior de um segmento s)

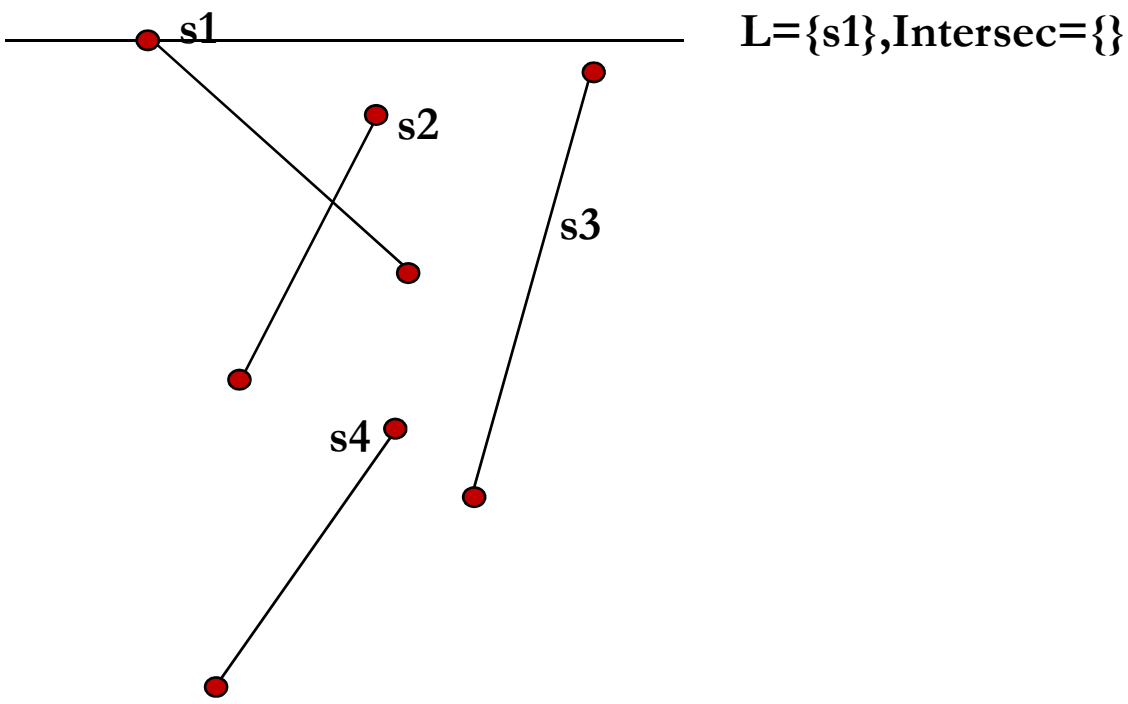
- Ações:

  1. inserir o segmento s no *status* da reta l
  2. Testar a interseção de s contra os segmentos no status de l

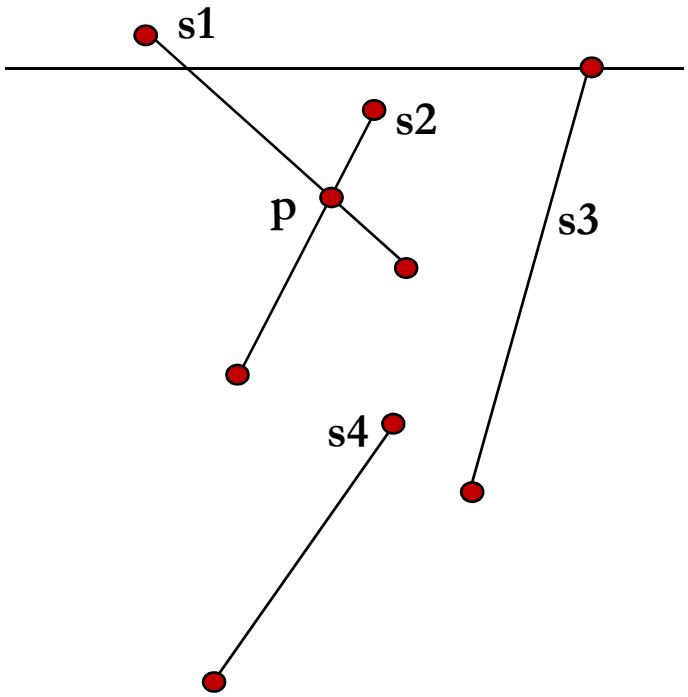
Problemas fundamentais: algoritmo para interseção entre *n* segmentos



Problemas fundamentais: algoritmo para interseção entre *n* segmentos

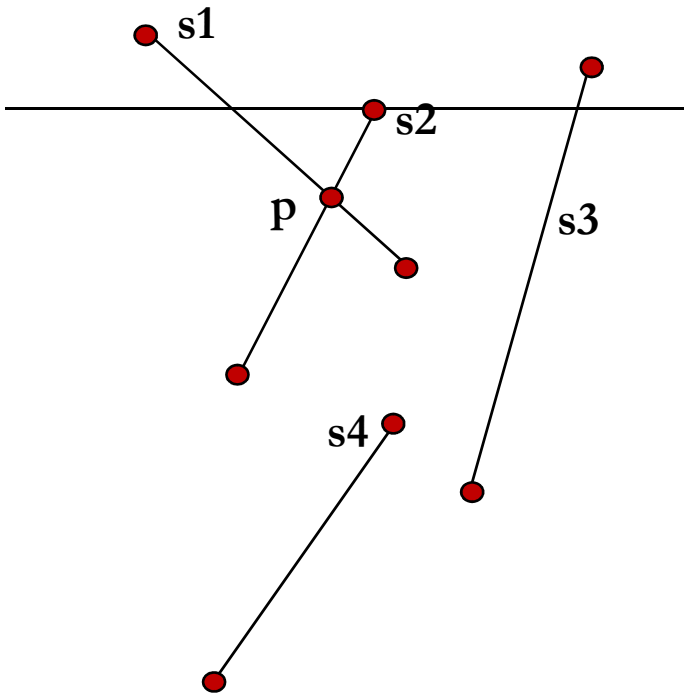


Problemas fundamentais: algoritmo para interseção entre *n* segmentos



L={s1,s3},Intersec={}

Problemas fundamentais: algoritmo para interseção entre *n* segmentos



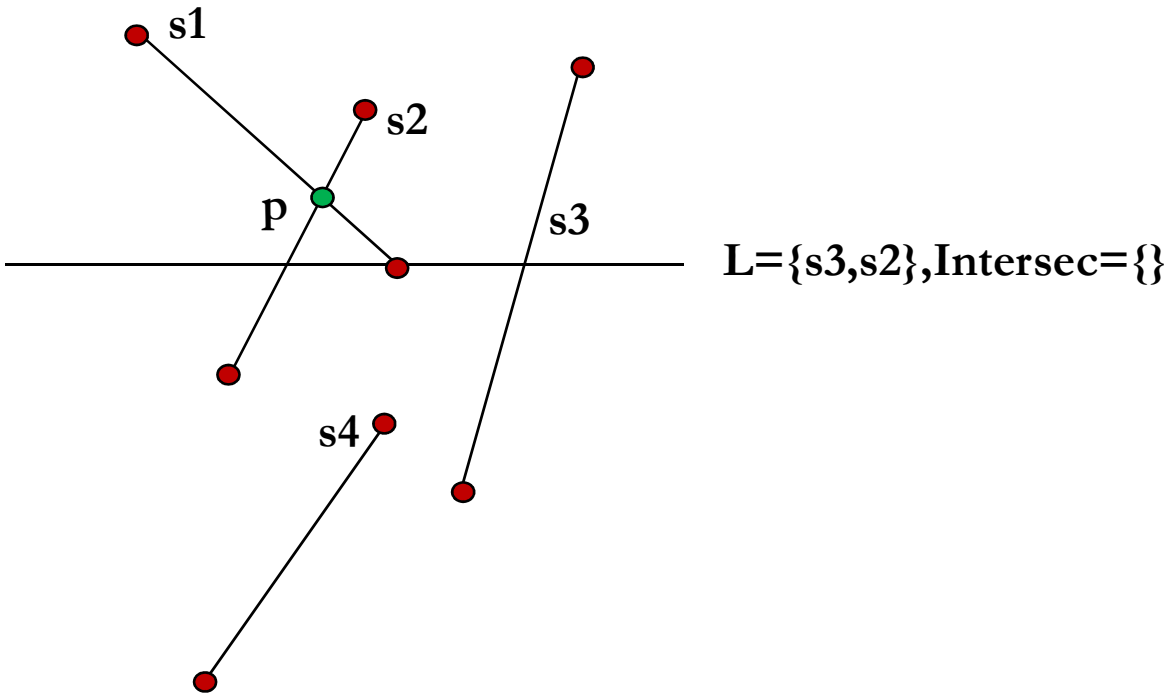
L={s1,s3,s2},Intersec={p}

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Caso 2 (o evento é um ponto extremo inferior de um segmento s)
- Ação: remover o segmento s no status da reta l

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

Por enquanto, ignoremos as interseções... O segmento s1 sai do estado de L



Problemas fundamentais: algoritmo para interseção entre *n* segmentos

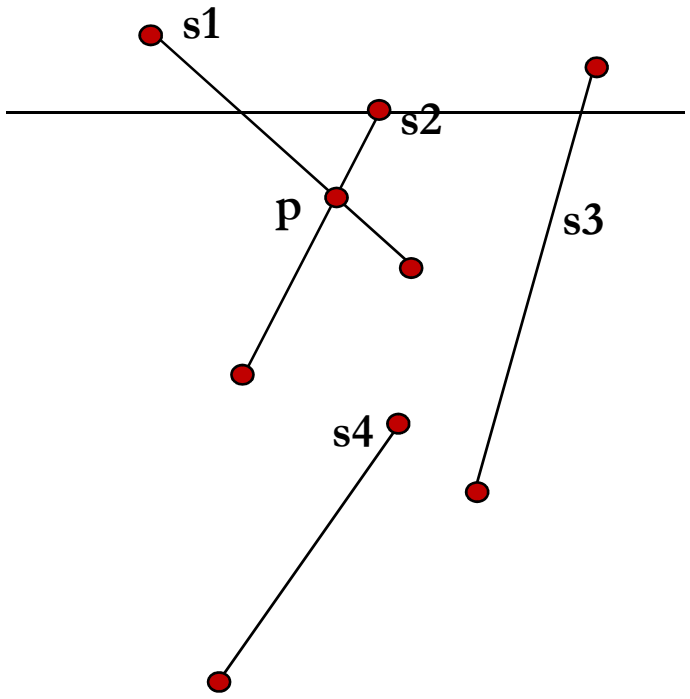
- Testamos apenas segmentos intersectados por l
- Mesmo assim o número de interseções pode ser quadrático
- Como evitar todas essas interseções?

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Muitos dos segmentos intersectados por l podem estar distantes entre si
- Ideia:
  a. Ordenar os segmentos da esquerda para direita na ordem das interseções para captura a noção de proximidade na horizontal
  b. Testar apenas os segmentos que são adjacentes horizontalmente

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

Quando o evento s2 é tratado verificamos a interseção apenas entre s1 e s2 e s2 e s3



L={s1,s2,s3},Intersec={p}

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Agora, para cada novo segmento que muda o status da reta de varredura testamos apenas um segmento a esquerda e outro a direita

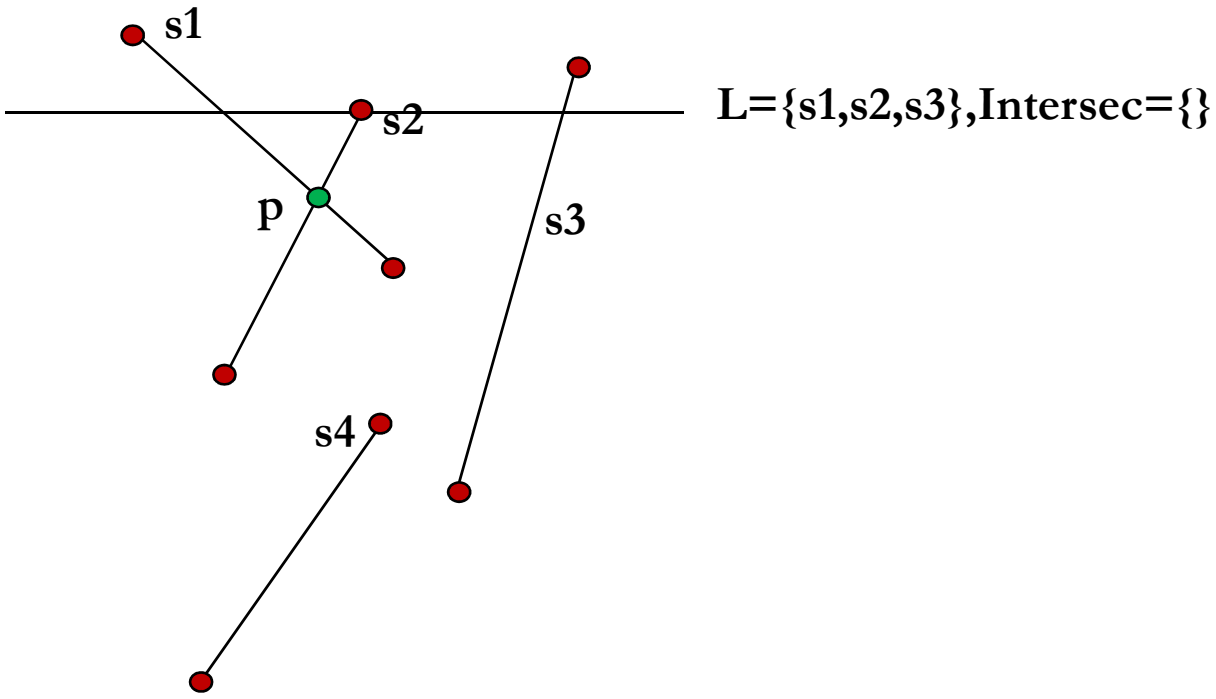
- Quando a reta se move para um novo status, um segmento pode passar a ter outros segmentos adjacentes

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

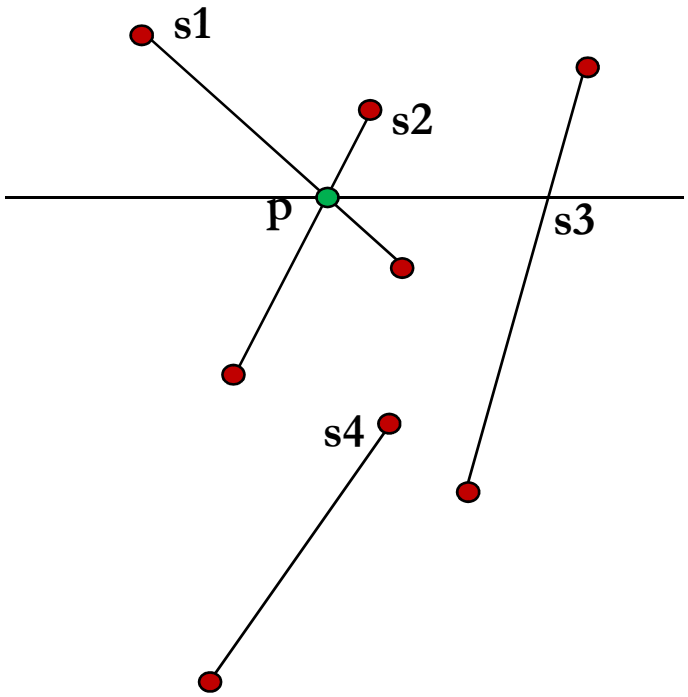
- Portanto, o status da reta l passa a ser a sequência de segmentos ordenados da esquerda para a direita que intersectam l

- Este novo status não muda mais somente nos extremos, mas também nos pontos de interseção

Problemas fundamentais: algoritmo para interseção entre *n* segmentos



Problemas fundamentais: algoritmo para interseção entre *n* segmentos



L={s2,s1,s3},Intersec={}

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Nos pontos de interseção a ordem dos segmentos muda localmente
- É preciso testar os segmentos que mudaram de ordem contra seus novos adjacentes

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Detalhes de implementação

Utiliza-se uma fila de prioridades Q para tratar os eventos
Um evento **e** tem prioridade sobre um evento **e'** se:

$e_y > e'_y$

a. $e_y = e'_y$ e $e_x < e'_x$

O evento é determinado por um ponto e os segmentos que nele começam

A busca e inserção de elementos na fila tem custo O(log m), onde m é o número de eventos (pontos extremos + interseções)

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Detalhes de implementação

Para tratar o status utiliza-se uma arvore binária de busca balanceada T

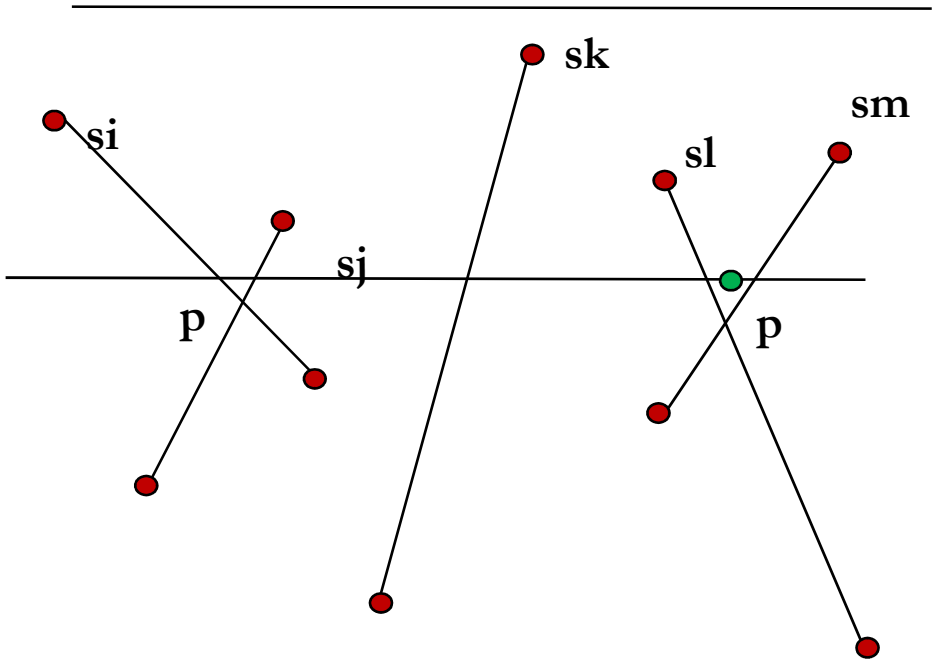
Os segmentos que intersectam l são armazenados nas folhas de T

Para auxiliar a busca, os nós internos armazenam os segmentos mais a direita na subárvore esquerda

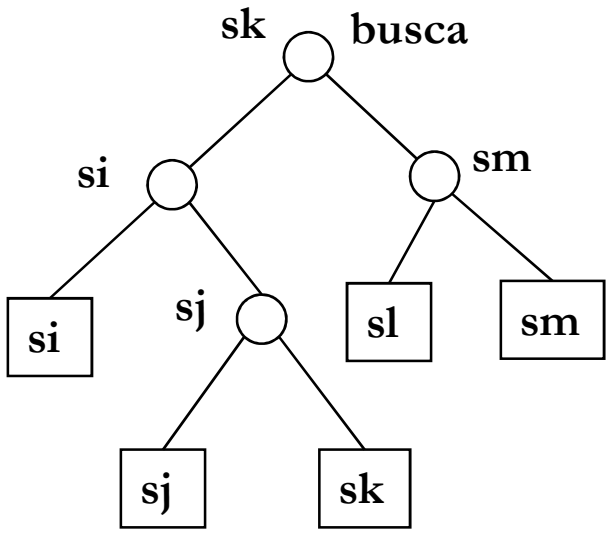
Problemas fundamentais: algoritmo para interseção entre *n* segmentos

- Teorema 5.4. Seja *S* um conjunto de *n* segmentos no plano. Todos os pontos de interseção dos segmentos em *S* podem ser identificados e reportados em tempo O($n \log n + I \log n$) e com complexidade de espaço O(n), onde I é o número de pontos de interseção

Problemas fundamentais: algoritmo para interseção entre *n* segmentos



T – árvore binária balanceada de busca



Em cada nó interno armazenamos o segmento mais a direita da subárvore esquerda. Os segmentos (dados) são armazenados nas folhas

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

Algorithm **FINDINTERSECTIONS(*S*)**
Input. A set *S* of line segments in the plane.
Output. The set of intersection points among the segments in *S*, with for each intersection point the segments that contain it.

1. Initialize an empty event queue Q. Next, insert the segment endpoints into Q; when an upper endpoint is inserted, the corresponding segment should be stored with it.
2. Initialize an empty status structure T.
3. **while** Q is not empty
4. **do** Determine the next event point *p* in Q and delete it.
5. HANDLEEVENTPOINT(*p*)

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

HANDLEEVENTPOINT(*p*)

1.Let $U(p)$ be the set of segments whose upper endpoint is p; these segments are stored with the event point p. (For horizontal segments, the upper endpoint is by definition the left endpoint.)

2.Find all segments stored in T that contain p; they are adjacent in T. Let $L(p)$ denote the subset of segments found whose lower endpoint is p, and let $C(p)$ denote the subset of segments found that contain p in their interior.

3.if $L(p) \cup U(p) \cup C(p)$ contains more than one segment

4. **then** Report p as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.

5.Delete the segments in $L(p) \cup C(p)$ from T.

6. Insert the segments in $U(p) \cup C(p)$ into T. The order of the segments in T should correspond to the order in which they are intersected by a sweep line just below p. If there is a horizontal segment, it comes last among all segments containing p.

7.(* Deleting and re-inserting the segments of $C(p)$ reverses their order. *)

8. **if** $U(p) \cup C(p) = 0$

9. **then** Let sl and sr be the left and right neighbors of p in T.

10. FINDNEWEVENT(sl,sr, p)

11. **else** Let s be the leftmost segment of $U(p) \cup C(p)$ in T.

12. Let sl be the left neighbor of s in T.

13. FINDNEWEVENT(sl,s , p)

14. Let s' be the rightmost segment of $U(p) \cup C(p)$ in T.

15. Let sr be the right neighbor of s' in T.

16. FINDNEWEVENT(s'',sr, p)

Problemas fundamentais: algoritmo para interseção entre *n* segmentos

FINDNEWEVENT(*sl*,*sr*, *p*)

1. **if** sl and sr intersect below the sweep line, or on it and to the right of the current event point p, and the intersection is not yet present as an event in Q

2. **then** Insert the intersection point as an event into Q.

Problemas fundamentais: par mais próximo

- Problema: dados *n* pontos no plano, encontrar o par mais próximo

- Aplicações: computação gráfica, visão computacional, GIS, modelagem de moléculas, controle de tráfego, etc.

Problemas fundamentais: par mais próximo

- Solução trivial: força bruta $O(n^2)$ (Por que?)

- Podemos resolver o par mais próximo sem testar todos os pares?

Problemas fundamentais: par mais próximo

- Abordagem incremental:

  1. Processa-se os pontos $p_1,...,p_n$, um de cada vez, na ordem em que são dados

  2. A cada passo identifica-se o par mais próximo dentre os pontos já vistos

  3. Ao final identifica-se o par mais próximo dentre todos os pares de pontos

Problemas fundamentais: par mais próximo

- Observe que em cada passo *k*, a nova estimativa de distância δ é atualizada somente se existem pontos p_j, dentro do círculo centrado em $B(p_K, \delta)$ de raio δ

- Neste caso, o novo par é formado por (p_k, p)

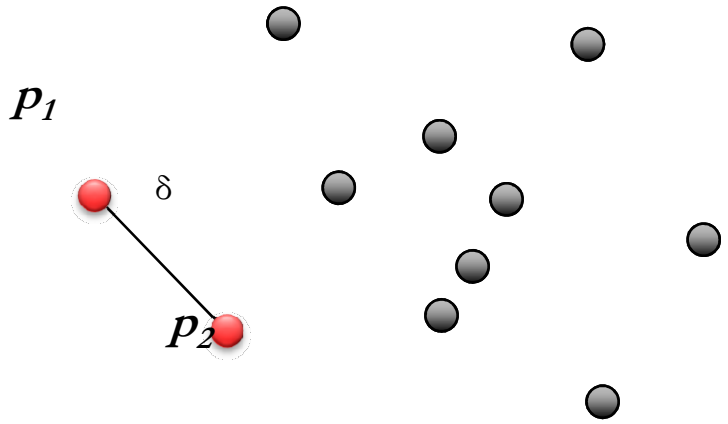
$$p = \underset{p_j \in B(p_k, \delta)}{\operatorname{argmin}} dist(p_k, p_j)$$

- Além disso, a nova estimativa de distância é dada por

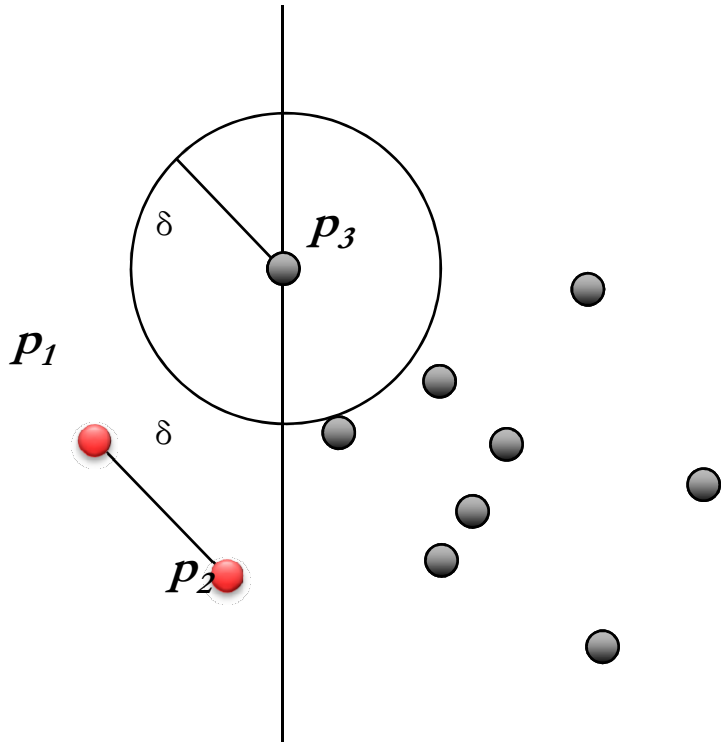
$$\delta = \min_{p_j \in B(p_k, \delta)} dist(p_k, p_j)$$

- Deste modo, a cada passo temos uma nova zona de influência dada por $B(p_K, \delta)$ que limita o número de testes

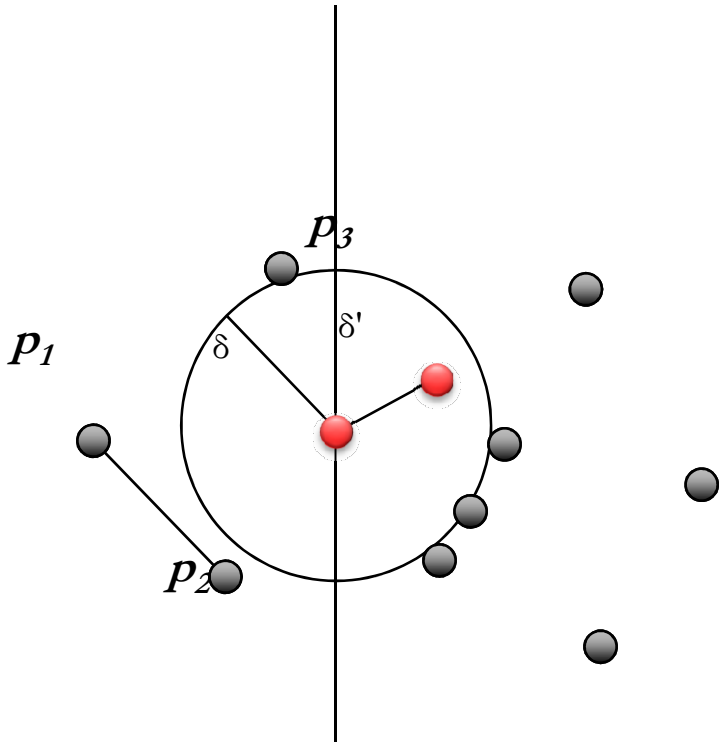
Problemas fundamentais: par mais próximo



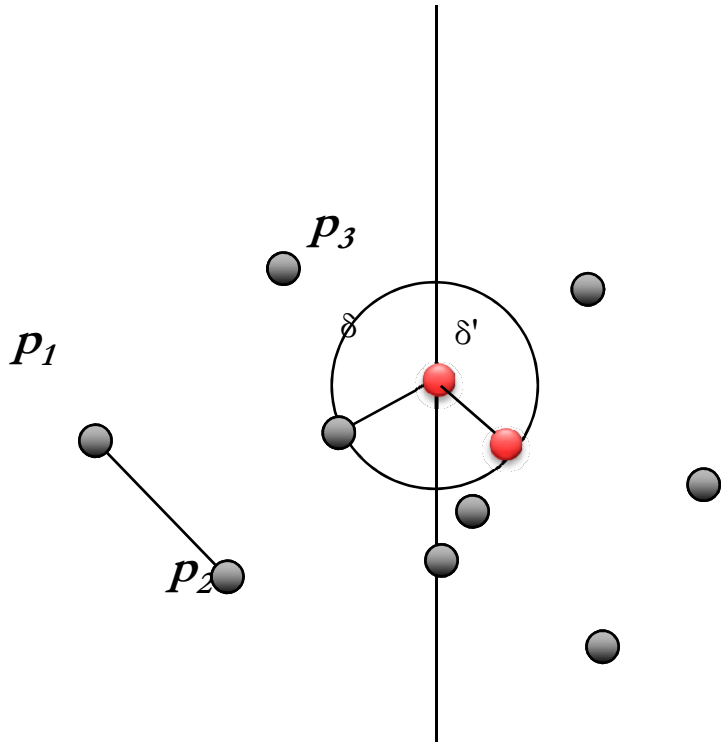
Problemas fundamentais: par mais próximo



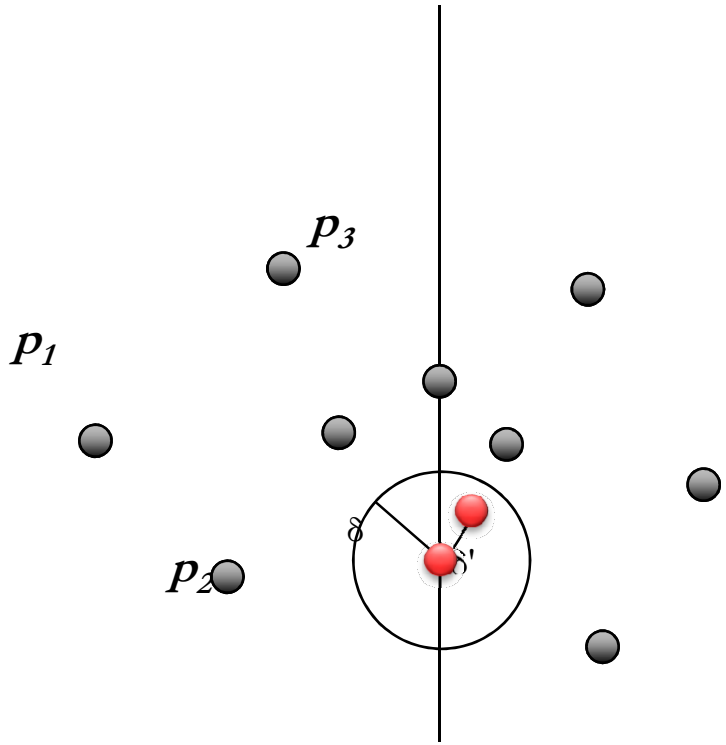
Problemas fundamentais: par mais próximo



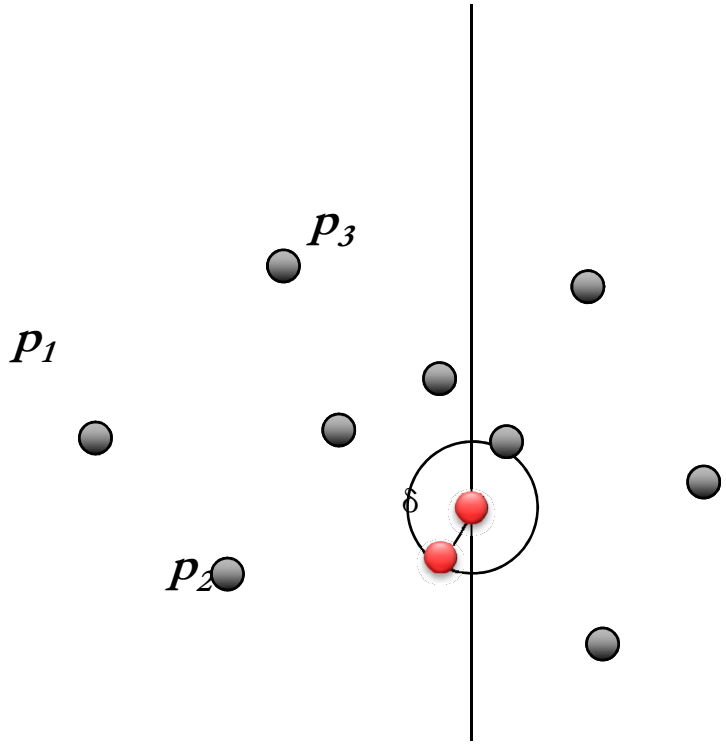
Problemas fundamentais: par mais próximo



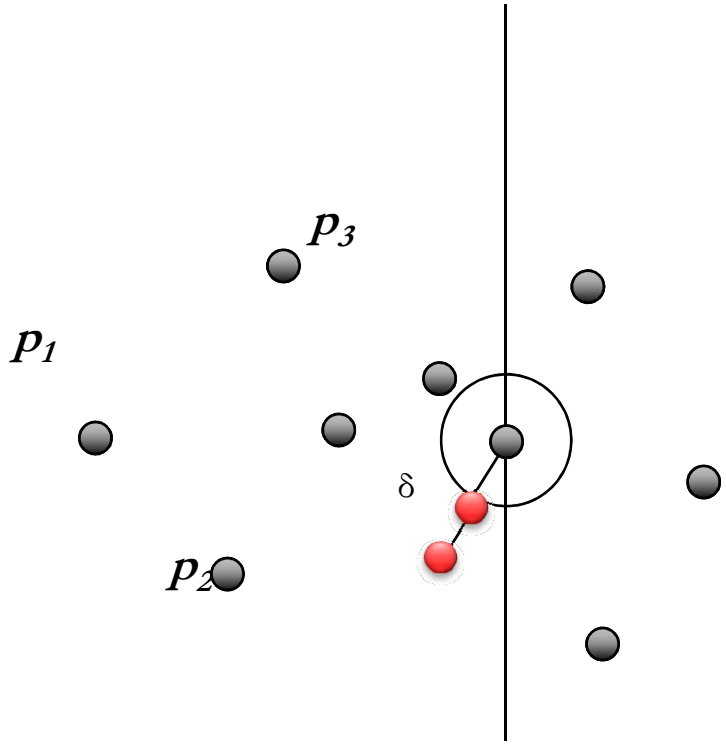
Problemas fundamentais: par mais próximo



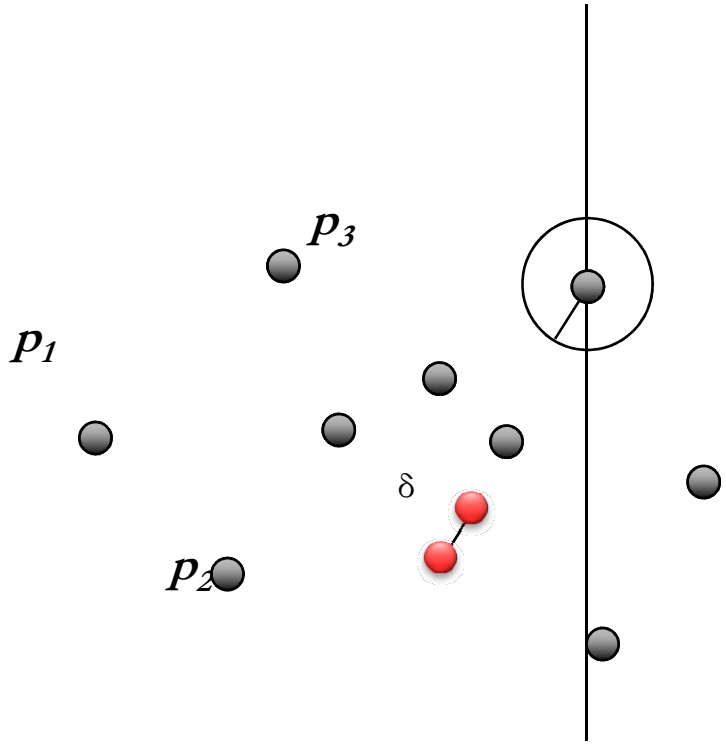
Problemas fundamentais: par mais próximo



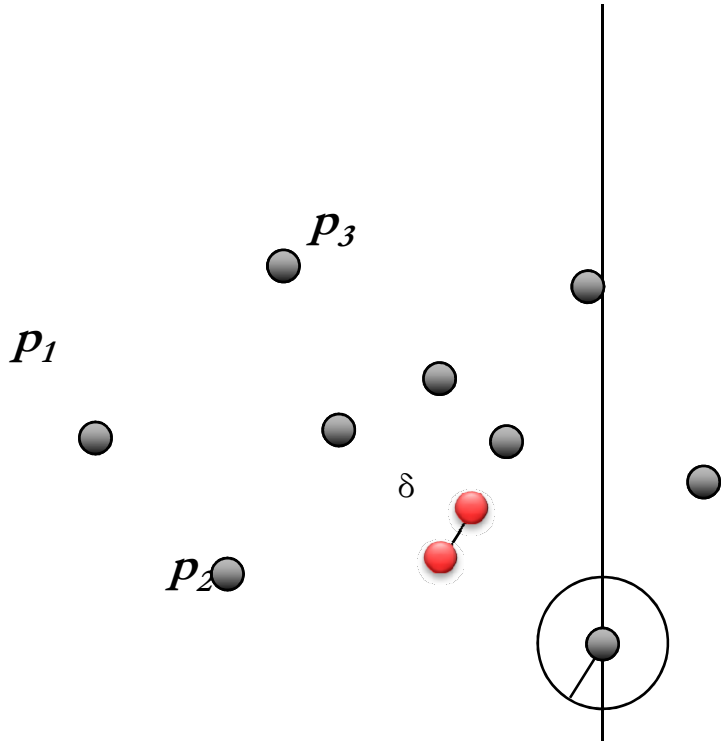
Problemas fundamentais: para mais próximo



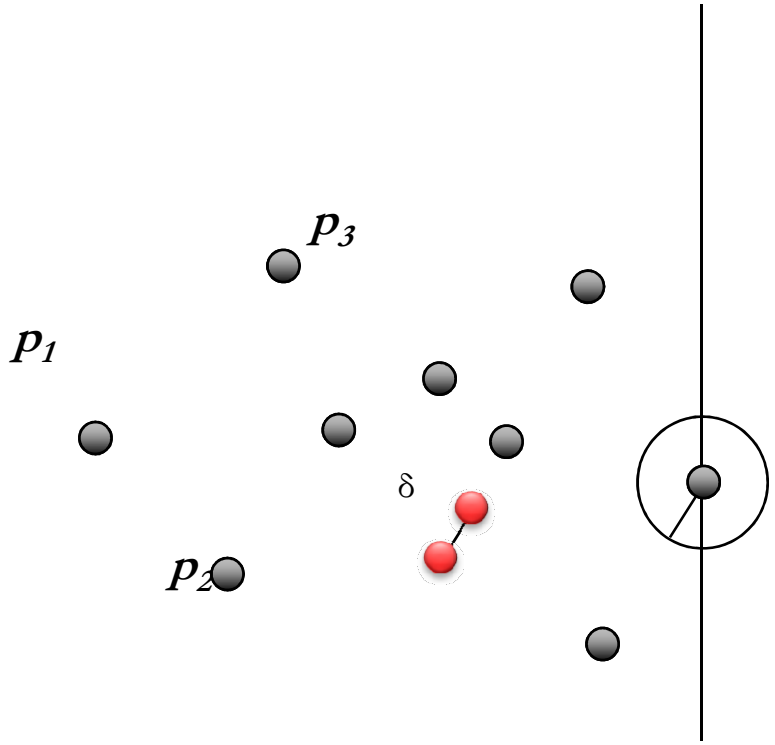
Problemas fundamentais: para mais próximo



Problemas fundamentais: para mais próximo



Problemas fundamentais: par mais próximo



Problemas fundamentais: par mais próximo

- É possível identificar as ideias intuitivas com o método de varredura do plano
- A linha de varredura é a linha vertical que passa por cada ponto, caminhando da esquerda para a direita
- Os eventos são determinados por cada ponto a ser processado
- O estado (status) de cada linha de varredura é a menor distância atual e o par mais próximo dentre os pontos já visitados

Problemas fundamentais: par mais próximo

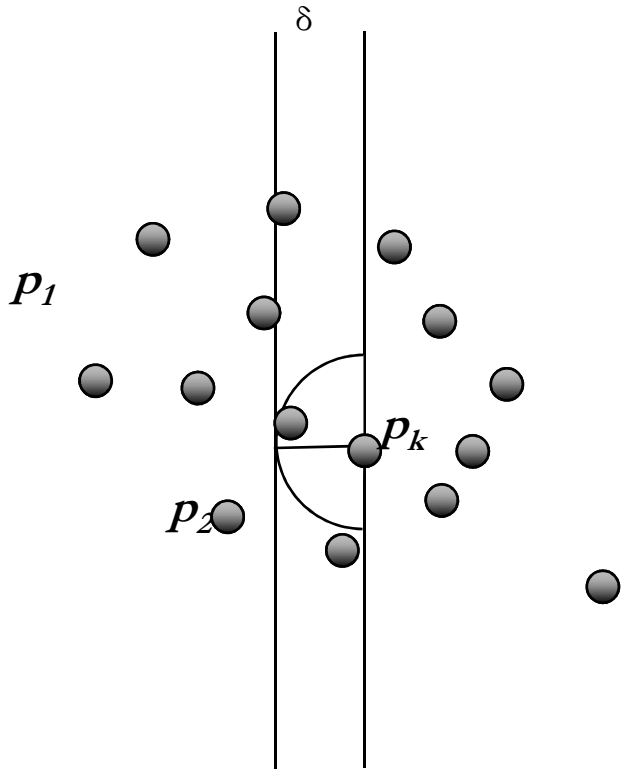
- Como determinar todos os pontos a serem testados a cada novo p_k na fila de eventos?

- Devemos ordenar todos os pontos em função da coordenada x

Problemas fundamentais: par mais próximo

- Deste modo podemos facilmente definir uma faixa vertical que restringe os pontos a serem testados

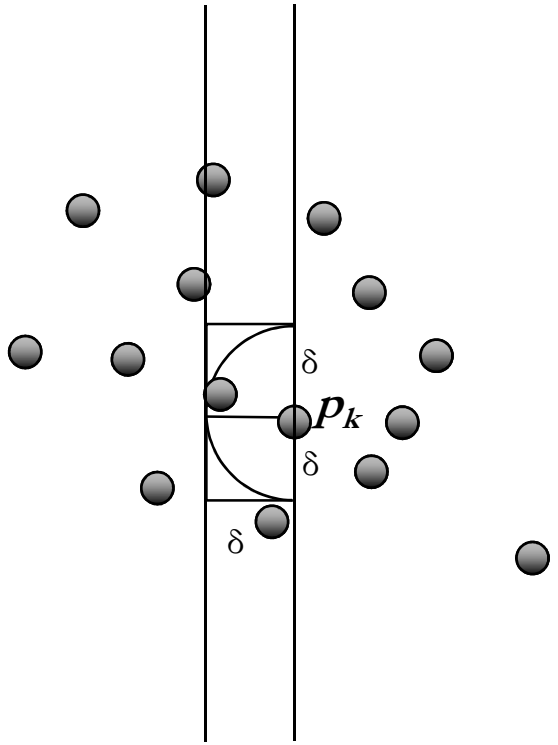
- Além disso, dentro da faixa vertical, apenas os pontos que estão a uma distância δ na coordenada y podem estar mais próximos de p_k que a estimativa atual



Problemas fundamentais: par mais próximo

•Finalmente, todos os candidatos a formarem um par mais próximo com p_k formam um retângulo de largura δ e altura 2δ

•Para efetuar uma busca eficiente mantemos os pontos ordenados dentro da faixa vertical ordenados pela coordenada y em uma lista L

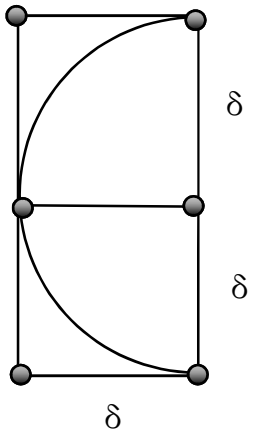


Problemas fundamentais: par mais próximo

- Por que tal estratégia é eficiente?
- Porque dentro de uma zona de candidatos limitamos o número de candidatos a uma constante

Problemas fundamentais: par mais próximo

- De fato no retângulo de largura δ e altura 2δ podemos ter no máximo 6 pontos



Problemas fundamentais: par mais próximo

Algorithm **IncrementalClosestPair(*S*)**

Input. A set S of points in the plane.

Output. A pair of points (p,p') and the minimum distance δ

1.Create an event queue sorting all points lexicographically (in x and then in y)

2.Star the sweeping at p_2 with $\delta=dist(p_1,p_2)$. The vertical strip covers initially the region bounded by $[x_1,x_2]$. The list L starts with p_1 and p_2 sorted by y

3.At step k, move the sweeping line to x_k and the vertical strip from $[x_i,x_k]$ to $[x_j,x_k]$, deleting all points p_u, $i<u<j$, such that $p_k-p_u \geq \delta$

4.Insert p_k in L and check whether there is a neighbor q of p_k in L such that $dist(p_k,q)<\delta$. If such point exists, update the closest pair and δ

Problemas fundamentais: par mais próximo

- Complexidade:
  - Para criar a fila de eventos no passo 1 precisamos ordenar os pontos o que custa O(n log n).
  - Para manter a lista L ordenada gastamos O(log n), por operação, usando uma árvore balanceada. Como o número de operações no passo 3 é n, temos o custo total O(n log n)
  - No passo 4, para incluir p_k em L e identificar cada vizinho gastamos O(log$n$). Como o número de candidatos é fixo temos o custo de O(n log n)(inserimo n pontos em L no total com custo O(log n)
  - O custo total somando todos custos em cada passo é O(n log n) e portanto o algoritmo é ótimo

Problemas fundamentais: círculo mínimo

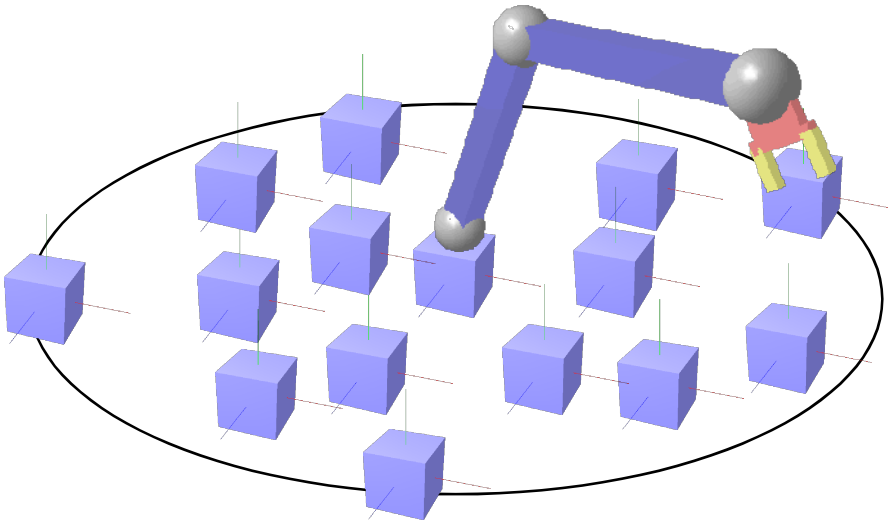
- Problema:

  Dados n pontos no plano encontrar o menor círculo que contém todos eles

- Consiste em determinar o centro do menor círculo que contém todos os seus pontos do conjunto e o seu raio

Problemas fundamentais: círculo mínimo

- Aplicações
  - Posicionar um hospital de forma a atender toda a comunidade em menor tempo
  - Planejar o centro de um bombardeio de forma mais eficiente
  - Posicionar um braço de robô no centro de um conjunto de objetos para minimizar o tempo para movimentá-los

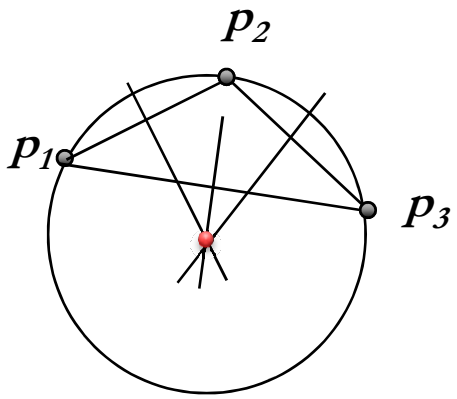


Problemas fundamentais: círculo mínimo

- Estudo da geometria do problema
  - Caso (n=1) – círculo mínimo com centro no único ponto e raio zero
  - Caso (n=2) – círculo mínimo com centro em $(p_1+p_2)/2$ e raio igual dist$(p_1,p_2)/2$

Problemas fundamentais: círculo mínimo

- Caso (n=3)
  - Três pontos colineares – supondo p_2 entre p_1 e p_3 então o círculo tem centro em $(p_1+p_3)/2$ e raio dist(p_1,p_3)
  - Três pontos em posição geral – consideremos o circuncírculo



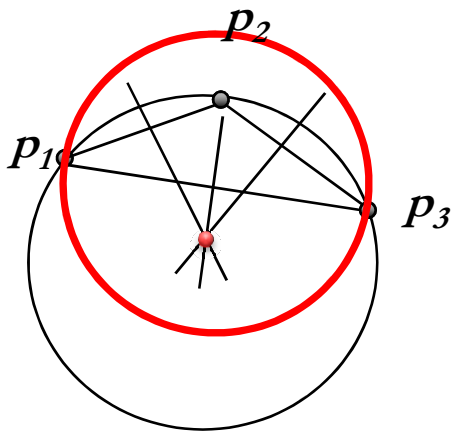
Problemas fundamentais: círculo mínimo

- Caso (n=3)

  •Três pontos em posição geral: consideremos o circuncírculo

  •O circuncírculo não necessariamente é o menor círculo

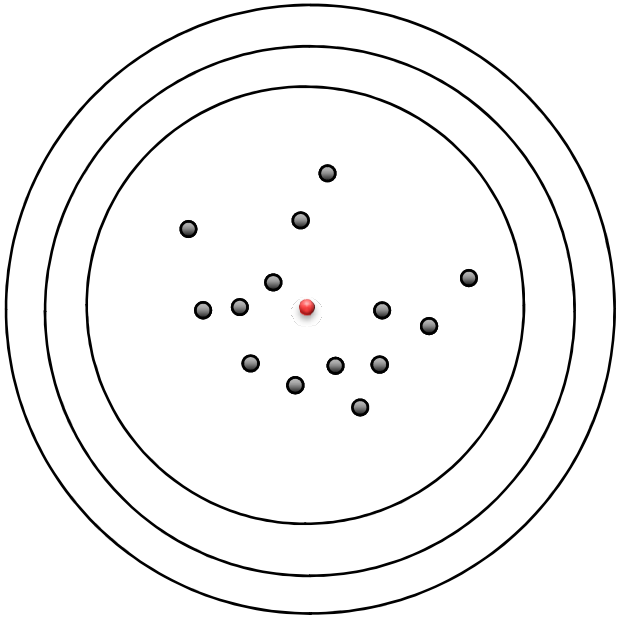
  •Se o triângulo contém um ângulo obtuso em p_2 então o diâmetro p_1p_3 determina um círculo menor que contêm p_2



Problemas fundamentais: círculo mínimo

- Caso (n≥3)

  •Sempre é possível encontrar um círculo que contenha todos os pontos

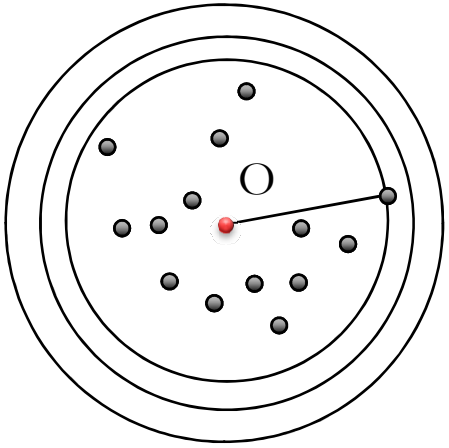


Problemas fundamentais: círculo mínimo

- Caso (n≥3)

  •Se ele não toca nenhum ponto de S então ele não pode ser mínimo.

  •É possível reduzir seu raio mantendo seu centro e reduzindo o raio até que toque o ponto mais distante de O

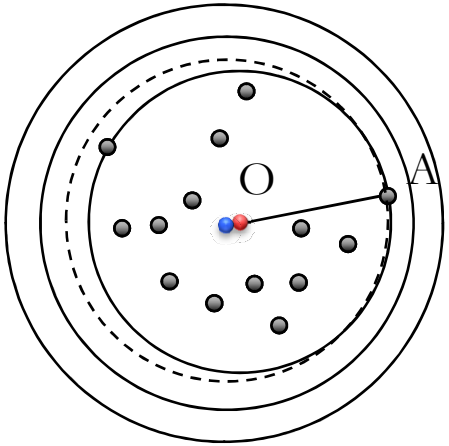


Problemas fundamentais: círculo mínimo

- Caso (n≥3)

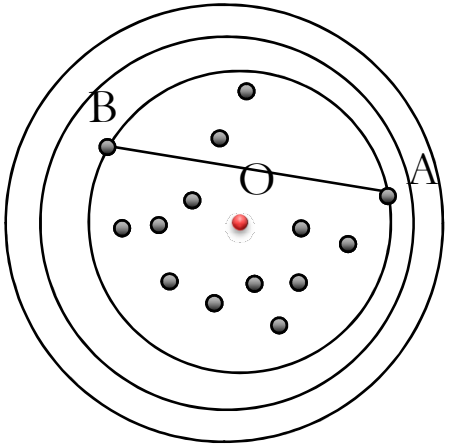
  •Se ele não toca pelo menos 2 pontos de S então ele não pode ser mínimo.

  •É possível reduzir seu raio mantendo mantendo o contato com o ponto de contato atual A e movendo o centro O na direção radial de A



Problemas fundamentais: círculo mínimo

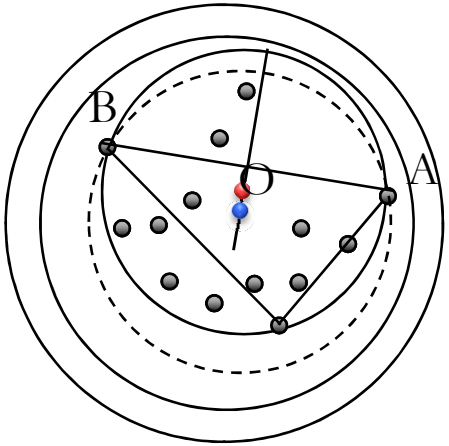
- Caso (n≥3)
  - Se AB é o diâmetro do círculo então achamos o mínimo



Problemas fundamentais: círculo mínimo

- Caso (n≥3)

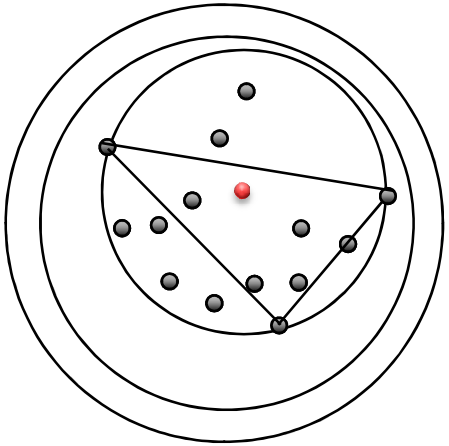
  •Senão, movemos o centro na direção da mediatriz de AB até encontrar um novo ponto de contato



Problemas fundamentais: círculo mínimo

- Caso (n≥3)

  •Resumo: para encontrar o círculo que contém S basta considerar todos os círculos definidos por dois ou três pontos de S que contenham todos os pontos de S



Problemas fundamentais: círculo mínimo

- Caso (n≥3)

  •Resumo: para encontrar o círculo que contém S basta considerar todos os círculos definidos por dois ou três pontos de S que contenham todos os pontos de S

  •Se o círculo tocar dois pontos que determinam seu diâmetro então ele é mínimo. Caso contrário, se deve existir mais um ponto de contato

  •Se três pontos de contato formam um ângulo obtuso então é possível reduzí-lo ainda mais (ver slide 90)

  •Se os três ângulos são agudos então o círculo correspondente é mínimo

Problemas fundamentais: círculo mínimo

Algorithm - MinimumEnclosingCircle(S)

Input. Set S of points in the plane

Output. Center c and radius r of the minimum enclosing circle

1.Draw a circle at center, c, such that points of given set lie within the circle. Clearly, this circle can be made smaller (or else we have the solution)

2.Make a circle smaller by finding the point A farthest from the center of the circle, and drawing a new circle with the same center and passing through the point A. These two steps produce a smaller enclosing circle. The reason that the new circle is smaller is that this new circle still contains all the points of given set, except now it passes through farthest point, x, rather than enclosing it.

3.If the circle passes through 2 or more points, proceed to step 4. Otherwise, make the circle smaller by moving the center towards point A, until the circle makes contact with another point B from the set.

4.At this stage, we have circle, C, that passes through two or more points of a given set. If the circle contains an interval (point-free interval) of arc greater than half the circle's circumference on which no points lie, the circle can be made smaller. Let D and E be the points at the ends of this point-free interval. While keeping D and E on the circle's boundary, reduce the diameter of the circle until we have either case (a) or case (b).

Problemas fundamentais: círculo mínimo

Case (a) The diameter is the distance DE.
We are done!

Case (b) The circle C touches another point from the set, F.
Check whether there exits point-free arc intervals of length more than half the circumference of C.
IF no such point-free arc intervals exit THEN
We are done!
Else
Goto step 4.
In this case, three points must lie on an arc less than half the circumference in length. We repeat step 4 on the outer two of the three points on the arc.

Problemas fundamentais: círculo mínimo

Algorithm **BRUTEFORCE(S) :**
Input. A set *S* of points in the plane.
Output. The smallest circle containing all points in S
1.Consider all pairs and triplets of S.
2.Choose the smallest circle defined by two or three points in S that contains all points in s

Complexidade: $n\left(\binom{n}{2}+\binom{n}{3}\right)=O(n^4)$

Problemas fundamentais: círculo mínimo

Algoritmo randômico baseado no seguinte Lema:

Seja $2<i<n$, e seja Si o conjunto com i pontos e Di o menor disco que contém os pontos de Si. Então temos que:

(i) Se $pi \in Di-1$ então $Di = Di-1$
(ii) Se $pi \notin Di-1$ então pi deve estar na fronteira de Di

Problemas fundamentais: círculo mínimo

Algorithm **MINIDISC(*S*)**
Input. A set *S* of n points in the plane.
Output. The smallest enclosing disc for S
1. Compute a random permutation p1,…,pn of S
2. Let D2 be the smallest enclosing disc for p1 and p2.
3. for i=3 to n
4. if pi∈Di-1 then
5. Di = Di-1
6. else
7. Di = **MINIDISCWITHPOINT({p1,…pi-1},pi)**
8. Return Dn

Problemas fundamentais: círculo mínimo

Algorithm **MINIDISCWITHPOINT(*S,q*)**
Input. A set *S* of n points in the plane and a point q such that there existis an enclosing disc for P with on its boundary.
Output. The smallest enclosing disc for S with q on its boundary

```
1.  Compute a random permutation p1,…,pn of S
2.  Let D1 be the smallest enclosing disc with p1 and q on its boundary.
3.  For j= 2 to n
4.      if pj∈Dj-1 then
5.          Dj = Dj-1
6.      else
7.          Dj = MINIDISCWITHTWOPOINTS({p1,…pj-1},pj,q)
8.  Return Dn
```

Problemas fundamentais: círculo mínimo

Algorithm **MINIDISCWITHTWOPOINTS(*S,q1,q2*)**
Input. A set *S* of n points in the plane with two points q1 and q2 such that there existis an enclosing disc for P with q1 and q2 on its boundary.
Output. The smallest enclosing disc for S with q1 and q2 on its boundary

```
1.  Compute a random permutation p1,…,pn of S
2.  Let D0 be the smallest enclosing disc with q1 and q2 on its boundary.
3.  for k=2 to n
4.      if pk∈Dk-1 then
5.          Dk = Dk-1
6.      else
7.          Dk = disc with q1,q2 and pk on its boundary
8.  Return Dn
```

Problemas fundamentais: círculo mínimo

Complexidade do algoritmo randômico:

$$O(n) + \sum_{i=2,n} O(i)\frac{2}{i} = O(n)$$

Por que???????

Problemas fundamentais: Referências

- Localização
  - ***Notas de aula*** **do Prof. Luiz Henrique Figueiredo – IMPA.**

- Interseção de Segmentos
  - **F. P. Preparata e M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, 1985.**
  - **M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 1997.**

- Par mais próximo
  - ***Notas de aula*** **do Prof. Luiz Henrique Figueiredo – IMPA.**
  - **F. P. Preparata e M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, 1985.**

- Minimum Enclosing Circle

  - ***Notas de aula*** **do Prof. Luiz Henrique Figueiredo – IMPA.**
  - http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/CG-Applets/Center/centercli.htm